(12) INTERNATIONAL APPLICATION PUBLISHED UNDER THE PATENT COOPERATION TREATY (PCT)

(19) World Intellectual Property Organization
International Bureau

(43) International Publication Date
30 November 2000 (30.11.2000)

PCT

(10) International Publication Number
WO 00/72170 A1

(51) International Patent Classification[7]: G06F 15/167

(21) International Application Number: PCT/US00/14282

(22) International Filing Date: 24 May 2000 (24.05.2000)

(25) Filing Language: English

(26) Publication Language: English

(30) Priority Data:
60/135,664 24 May 1999 (24.05.1999) US
60/154,150 15 September 1999 (15.09.1999) US

(71) Applicants *(for all designated States except US)*: **HEWLETT-PACKARD COMPANY** [US/US]; Intellectual Property Administration, P.O. Box 272400, 3404 East Harmony Road, M/S 35, Fort Collins, CO 80527-2400 (US). **IBM CORPORATION** [US/US]; Intellectual Property Administration, P.O. Box 272400, 3404 East Harmony Road, M/S 35, Fort Collins, CO 80527-2400 (US). **COMPAQ COMPUTER CORP.** [US/US]; Intellectual Property Administration, P.O. Box 272400, 3404 East Harmony Road, M/S 35, Fort Collins, CO 80527-2400 (US). **ADAPTEC, INC.** [US/US]; Intellectual Property Administration, P.O. Box 272400, 3404 East Harmony Road, M/S 35, Fort Collins, CO 80527-2400 (US).

(72) Inventors; and

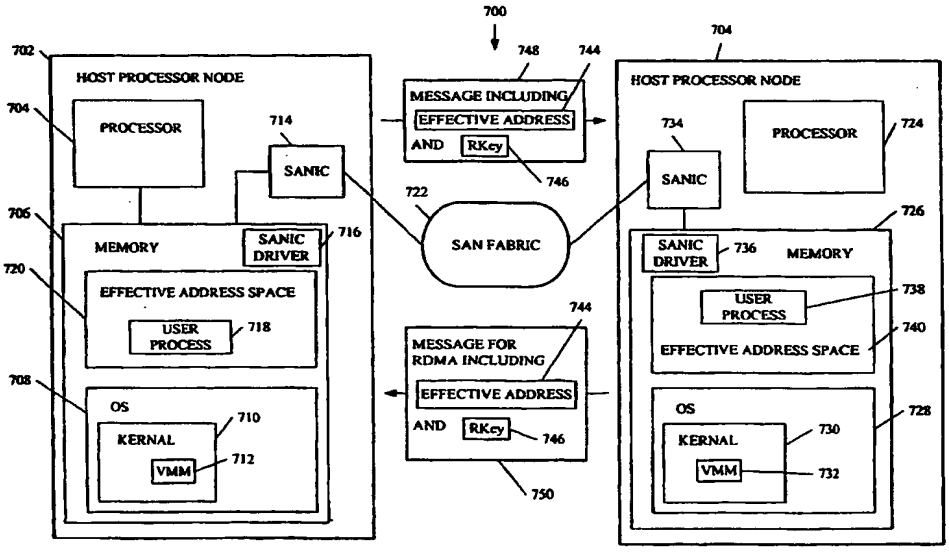
(75) Inventors/Applicants *(for US only)*: **KRAUSE, Michael** [US/US]; 17523 Bear Creek Road, Boulder Creek, CA 95006 (US). **BENNER, Alan, F.** [US/US]; 1 Ridgeview Road, Poughkeepsie, NY 12603 (US). **GARCIA, David, J.** [US/US]; 24100 Hutchinson Road, Los Gatos, CA 95033 (US). **JOHN RECIO, Renato** [US/US]; 6707 Winnepeg Cove, Austin, TX 78759 (US).

(74) Agents: **BILLIG, Patrick** et al.; Hewlett-Packard Company, Intellectual Property Administration, P.O. Box 272400, 3404 E. Harmony Road, M/S 35, Fort Collins, CO 80527-2400 (US).

(81) Designated States *(national)*: AE, AL, AM, AT, AU, AZ, BA, BB, BG, BR, BY, CA, CH, CN, CR, CU, CZ, DE, DK, DM, EE, ES, FI, GB, GD, GE, GH, GM, HU, ID, IL, IN, IS, JP, KE, KG, KP, KR, KZ, LC, LK, LR, LS, LT, LU, LV,

[Continued on next page]

(54) Title: MEMORY MANAGEMENT IN DISTRIBUTED COMPUTER SYSTEM



(57) Abstract: A method of managing memory in a distributed computer system (700) includes binding a remote key (746) to a first address (744) representing a contiguous memory address range at a first endnode (702). The bound remote key (746) and first address (744) are sent from the first endnode (702) to a second endnode (704) on a communication fabric (722). A remote direct memory access operation (750) is performed from the second endnode (704) to access the contiguous memory address range including sending the bound remote key (746) and the first address (744) from the second endnode (704) to the first endnode (702) on the communication fabric (722).

WO 00/72170 A1

MEMORY MANAGEMENT IN DISTRIBUTED COMPUTER SYSTEM

The Field of the Invention

The present invention generally relates to memory management in computer systems, and more particularly to allocation of memory in a distributed computer system including local memory access protection and remote memory access protection.

Background of the Invention

Conventional computer systems typically employ region-based memory management techniques. In the region-based memory management technique, a large portion of memory (e.g., one megabyte) is allocated to an application. This large allocated memory portion is referred to as the application's memory domain. Conventional computer systems allocate the large portion of memory for the application by trapping to an operating system kernel process. The trap to the operating system kernel process is performed to change memory access permissions and the like.

Conventional computer systems employ protection tables storing information related to memory page access rights, and page tables storing information for mapping virtual to physical addresses.

These conventional memory management techniques are cumbersome when multiple applications share memory and require access to the same shared memory segments. For example, when one application accesses a shared memory segment, and then another application uses the same shared memory segment, the virtual to physical mappings in the page table change. These virtual to physical mapping changes are difficult to track by the multiple applications sharing the same memory segment.

For reasons stated above and for other reasons presented in greater detail in the Description of the Preferred Embodiment section of the present

specification, there is a need for an improved memory management technique for permitting memory access protection at a lower level granularity then currently employed by conventional memory management techniques which always allocate a large portion of memory to an application. In addition, an improved memory management technique is needed which efficiently supports shared memory. Moreover, the improved memory management technique should permit virtual addresses to be used by applications in a shared memory system so that the applications do not need to deal with physical addresses. Furthermore, an improved memory management technique is desired which permits some types of memory allocation without context switching and without trapping to an operating system kernel process.

Summary of the Invention

The present invention provides a method of managing memory in a distributed computer system. The method comprises binding a remote key to a first address representing a contiguous memory address range at a first endnode. The bound remote key and first address are sent from the first endnode to a second endnode on a communication fabric. A remote direct memory access operation is performed from the second endnode to access the contiguous memory address range including sending the bound remote key and the first address from the second endnode to the first endnode on the communication fabric.

PATENT COOPERATION TREATY

PCT

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

(PCT Article 36 and Rule 70)

Applicant's or agent's file reference 10002166-1	**FOR FURTHER ACTION**	See Notification of Transmittal of International Preliminary Examination Report (Form PCT/IPEA/416)
International application No. PCT/US00/14282	International filing date *(day/month/year)* 24 MAY 2000	Priority date *(day/month/year)* 24 MAY 1999
International Patent Classification (IPC) or national classification and IPC IPC(7): G06F 15/167 and US Cl.: 709/212		
Applicant HEWLETT-PACKARD COMPANY		

1. This international preliminary examination report has been prepared by this International Preliminary Examining Authority and is transmitted to the applicant according to Article 36.

2. This REPORT consists of a total of 3 sheets.

   ☐ This report is also accompanied by ANNEXES, i.e., sheets of the description, claims and/or drawings which have been amended and are the basis for this report and/or sheets containing rectifications made before this Authority. (see Rule 70.16 and Section 607 of the Administrative Instructions under the PCT).

   These annexes consist of a total of 0 sheets.

3. This report contains indications relating to the following items:

   I ☒ Basis of the report

   II ☐ Priority

   III ☐ Non-establishment of report with regard to novelty, inventive step or industrial applicability

   IV ☐ Lack of unity of invention

   V ☒ Reasoned statement under Article 35(2) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement

   VI ☐ Certain documents cited

   VII ☐ Certain defects in the international application

   VIII ☐ Certain observations on the international application

Date of submission of the demand	Date of completion of this report
22 DECEMBER 2000	15 MAY 2001
Name and mailing address of the IPEA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer LE LUU Peggy Harrod Telephone No. (703) 305-3900

Form PCT/IPEA/409 (cover sheet) (July 1998) ★

INTERNATIONAL PRELIMINARY EXAMINATION REPORT

International application No. PCT/US00/14282

I. Basis of the report

1. With regard to the **elements** of the international application:*

[X] the international application as originally filed

[X] the description:
pages 1-39, as originally filed
pages NONE, filed with the demand
pages NONE, filed with the letter of

[X] the claims:
pages 40, as originally filed
pages NONE, as amended (together with any statement) under Article 19
pages NONE, filed with the demand
pages NONE, filed with the letter of

[X] the drawings:
pages 1-15, as originally filed
pages NONE, filed with the demand
pages NONE, filed with the letter of

[X] the sequence listing part of the description:
pages NONE, as originally filed
pages NONE, filed with the demand
pages NONE, filed with the letter of

2. With regard to the **language**, all the elements marked above were available or furnished to this Authority in the language in which the international application was filed, unless otherwise indicated under this item.
These elements were available or furnished to this Authority in the following language _____ which is:

[ ] the language of a translation furnished for the purposes of international search (under Rule 23.1(b)).

[ ] the language of publication of the international application (under Rule 48.3(b)).

[ ] the language of the translation furnished for the purposes of international preliminary examination (under Rules 55.2 and/or 55.3).

3. With regard to any **nucleotide and/or amino acid sequence** disclosed in the international application, the international preliminary examination was carried out on the basis of the sequence listing:

[ ] contained in the international application in printed form.

[ ] filed together with the international application in computer readable form.

[ ] furnished subsequently to this Authority in written form.

[ ] furnished subsequently to this Authority in computer readable form.

[ ] The statement that the subsequently furnished written sequence listing does not go beyond the disclosure in the international application as filed has been furnished.

[ ] The statement that the information recorded in computer readable form is identical to the writen sequence listing has been furnished.

4. [X] The amendments have resulted in the cancellation of:

[X] the description, pages NONE

[X] the claims, Nos. NONE

[X] the drawings, sheets/~~fig~~ NONE

5. [ ] This report has been drawn as if (some of) the amendments had not been made, since they have been considered to go beyond the disclosure as filed, as indicated in the Supplemental Box (Rule 70.2(c)).**

** Replacement sheets which have been furnished to the receiving Office in response to an invitation under Article 14 are referred to in this report as "originally filed" and are not annexed to this report since they do not contain amendments (Rules 70.16 and 70.17).*

***Any replacement sheet containing such amendments must be referred to under item 1 and annexed to this report.*

INTERNATIONAL PRELIM[redacted]RY EXAMINATION REPORT

Inte[redacted]nal application No.
PC[redacted]US00/14282

V. Reasoned statement under Article 35(2) with regard to novelty, inventive step or industrial applicability; citations and explanations supporting such statement

1. statement

Novelty (N)	Claims	1	YES
	Claims	NONE	NO
Inventive Step (IS)	Claims	1	YES
	Claims	NONE	NO
Industrial Applicability (IA)	Claims	1	YES
	Claims	NONE	NO

2. citations and explanations (Rule 70.7)

Claim 1 meets the criteria set out in PCT Article 33(2)-(4), because the prior art does not teach or fairly suggest the management of memory through the use of direct memory access operations by binding a (remote) key to a first address which represents a contiguous memory address range at a local node, transferring the key and address to a remote node, and using the key and address to allow a remote machine to conduct direct memory access of memory at the local node.

---------- NEW CITATIONS ----------

NONE

Brief Description of the Drawings

Figure 1 is a diagram of a distributed computer system for implementing the present invention.

Figure 2 is a diagram of an example host processor node for the computer system of Figure 1.

Figure 3 is a diagram of a portion of a distributed computer system employing a reliable connection service to communicate between distributed processes.

Figure 4 is a diagram of a portion of distributed computer system employing a reliable datagram service to communicate between distributed processes.

Figure 5 is a diagram of an example host processor node for operation in a distributed computer system implementing the present invention.

Figure 6 is a diagram of a portion of a distributed computer system illustrating subnets in the distributed computer system.

Figure 7 is a diagram of a switch for use in a distributed computer system implemented the present invention.

Figure 8 is a diagram of a portion of a distributed computer system.

Figure 9A is a diagram of a work queue element (WQE) for operation in the distributed computer system of Figure 8.

Figure 9B is a diagram of the packetization process of a message created by the WQE of Figure 9A into frames and flits.

Figure 10A is a diagram of a message being transmitted with a reliable transport service illustrating frame transactions.

Figure 10B is a diagram illustrating a reliable transport service illustrating flit transactions associated with the frame transactions of Figure 10A.

Figure 11 is a diagram of a layered architecture for implementing the present invention.

Figure 12 is a diagram of a computer system for illustrating memory access protection mechanisms.

Figure 13 is a diagram of an example effective address.

Figure 14 is a diagram of an example segment table entry.

Figure 15 is a diagram of an example virtual address.

Figure 16 is a diagram of an example page table entry.

Figure 17 is flow diagram of a mechanism for managing local memory access protection.

Figure 18 is a diagram of a distributed computer system having an embodiment of a mechanism for managing remote memory access protection according to the present invention.

Description of the Preferred Embodiments

In the following detailed description of the preferred embodiments, reference is made to the accompanying drawings which form a part hereof, and in which is shown by way of illustration specific embodiments in which the invention may be practiced. It is to be understood that other embodiments may be utilized and structural or logical changes may be made without departing from the scope of the present invention. The following detailed description, therefore, is not to be taken in a limiting sense, and the scope of the present invention is defined by the appended claims.

One embodiment of the present invention is directed to a memory management technique employing remote keys to allocate a small region of memory referred to as a memory window without trapping to an operating system kernel process. In such a memory management technique according to the present invention, a range of memory in the memory window or the entire memory window can be allocated by binding a remote key to the range of memory in the memory window or the entire memory window to permit memory access protection at a byte level granularity. One embodiment of a memory management technique according to the present invention provides an extremely light-weight technique for supporting shared memory. One embodiment of a memory management technique according to the present invention permits

remote memory management where user applications only deal with virtual addresses.

An example embodiment of a distributed computer system is illustrated generally at 30 in Figure 1. Distributed computer system 30 is provided merely for illustrative purposes, and the embodiments of the present invention described below can be implemented on computer systems of numerous other types and configurations. For example, computer systems implementing the present invention can range from a small server with one processor and a few input/output (I/O) adapters to massively parallel supercomputer systems with hundreds or thousands of processors and thousands of I/O adapters. Furthermore, the present invention can be implemented in an infrastructure of remote computer systems connected by an internet or intranet.

Distributed computer system 30 includes a system area network (SAN) 32 which is a high-bandwidth, low-latency network interconnecting nodes within distributed computer system 30. A node is herein defined to be any device attached to one or more links of a network and forming the origin and/or destination of messages within the network. In the example distributed computer system 30, nodes include host processors 34a-34d; redundant array independent disk (RAID) subsystem 33; and I/O adapters 35a and 35b. The nodes illustrated in Figure 1 are for illustrative purposes only, as SAN 32 can connect any number and any type of independent processor nodes, I/O adapter nodes, and I/O device nodes. Any one of the nodes can function as an endnode, which is herein defined to be a device that originates or finally consumes messages or frames in the distributed computer system.

A message is herein defined to be an application-defined unit of data exchange, which is a primitive unit of communication between cooperating sequential processes. A frame is herein defined to be one unit of data encapsulated by a physical network protocol header and/or trailer. The header generally provides control and routing information for directing the frame through SAN 32. The trailer generally contains control and cyclic redundancy check (CRC) data for ensuring packets are not delivered with corrupted contents.

SAN 32 is the communications and management infrastructure supporting both I/O and interprocess communication (IPC) within distributed computer system 30. SAN 32 includes a switched communications fabric (SAN FABRIC) allowing many devices to concurrently transfer data with high-bandwidth and low latency in a secure, remotely managed environment. Endnodes can communicate over multiple ports and utilize multiple paths through the SAN fabric. The multiple ports and paths through SAN 32 can be employed for fault tolerance and increased bandwidth data transfers.

SAN 32 includes switches 36 and routers 38. A switch is herein defined to be a device that connects multiple links 40 together and allows routing of frames from one link 40 to another link 40 within a subnet using a small header destination ID field. A router is herein defined to be a device that connects multiple links 40 together and is capable of routing frames from one link 40 in a first subnet to another link 40 in a second subnet using a large header destination address or source address.

In one embodiment, a link 40 is a full duplex channel between any two network fabric elements, such as endnodes, switches 36, or routers 38. Example suitable links 40 include, but are not limited to, copper cables, optical cables, and printed circuit copper traces on backplanes and printed circuit boards.

Endnodes, such as host processor endnodes 34 and I/O adapter endnodes 35, generate request frames and return acknowledgment frames. By contrast, switches 36 and routers 38 do not generate and consume frames. Switches 36 and routers 38 simply pass frames along. In the case of switches 36, the frames are passed along unmodified. For routers 38, the network header is modified slightly when the frame is routed. Endnodes, switches 36, and routers 38 are collectively referred to as end stations.

In distributed computer system 30, host processor nodes 34a-34d and RAID subsystem node 33 include at least one system area network interface controller (SANIC) 42. In one embodiment, each SANIC 42 is an endpoint that implements the SAN 32 interface in sufficient detail to source or sink frames transmitted on the SAN fabric. The SANICs 42 provide an interface to the host

processors and I/O devices. In one embodiment the SANIC is implemented in hardware. In this SANIC hardware implementation, the SANIC hardware offloads much of CPU and I/O adapter communication overhead. This hardware implementation of the SANIC also permits multiple concurrent communications over a switched network without the traditional overhead associated with communicating protocols. In one embodiment, SAN 32 provides the I/O and IPC clients of distributed computer system 30 zero processor-copy data transfers without involving the operating system kernel process, and employs hardware to provide reliable, fault tolerant communications.

As indicated in Figure 1, router 38 is coupled to wide area network (WAN) and/or local area network (LAN) connections to other hosts or other routers 38.

The host processors 34a-34d include central processing units (CPUs) 44 and memory 46.

I/O adapters 35a and 35b include an I/O adapter backplane 48 and multiple I/O adapter cards 50. Example adapter cards 50 illustrated in Figure 1 include an SCSI adapter card; an adapter card to fiber channel hub and FC-AL devices; an Ethernet adapter card; and a graphics adapter card. Any known type of adapter card can be implemented. I/O adapters 35a and 35b also include a switch 36 in the I/O adapter backplane 48 to couple the adapter cards 50 to the SAN 32 fabric.

RAID subsystem 33 includes a microprocessor 52, memory 54, read/write circuitry 56, and multiple redundant storage disks 58.

SAN 32 handles data communications for I/O and IPC in distributed computer system 30. SAN 32 supports high-bandwidth and scalability required for I/O and also supports the extremely low latency and low CPU overhead required for IPC. User clients can bypass the operating system kernel process and directly access network communication hardware, such as SANICs 42 which enable efficient message passing protocols. SAN 32 is suited to current computing models and is a building block for new forms of I/O and computer cluster communication. SAN 32 allows I/O adapter nodes to communicate

among themselves or communicate with any or all of the processor nodes in distributed computer system 30. With an I/O adapter attached to SAN 32, the resulting I/O adapter node has substantially the same communication capability as any processor node in distributed computer system 30.

Channel and Memory Semantics

In one embodiment, SAN 32 supports channel semantics and memory semantics. Channel semantics is sometimes referred to as send/receive or push communication operations, and is the type of communications employed in a traditional I/O channel where a source device pushes data and a destination device determines the final destination of the data. In channel semantics, the frame transmitted from a source process specifies a destination processes' communication port, but does not specify where in the destination processes' memory space the frame will be written. Thus, in channel semantics, the destination process pre-allocates where to place the transmitted data.

In memory semantics, a source process directly reads or writes the virtual address space of a remote node destination process. The remote destination process need only communicate the location of a buffer for data, and does not need to be involved with the transfer of any data. Thus, in memory semantics, a source process sends a data frame containing the destination buffer memory address of the destination process. In memory semantics, the destination process previously grants permission for the source process to access its memory.

Channel semantics and memory semantics are typically both necessary for I/O and IPC. A typical I/O operation employs a combination of channel and memory semantics. In an illustrative example I/O operation of distributed computer system 30, host processor 34a initiates an I/O operation by using channel semantics to send a disk write command to I/O adapter 35b. I/O adapter 35b examines the command and uses memory semantics to read the data buffer directly from the memory space of host processor 34a. After the data buffer is read, I/O adapter 35b employs channel semantics to push an I/O completion message back to host processor 34a.

In one embodiment, distributed computer system 30 performs operations that employ virtual addresses and virtual memory protection mechanisms to ensure correct and proper access to all memory. In one embodiment, applications running in distributed computed system 30 are not required to use physical addressing for any operations.

Queue Pairs

An example host processor node 34 is generally illustrated in Figure 2. Host processor node 34 includes a process A indicated at 60 and a process B indicated at 62. Host processor node 34 includes SANIC 42. Host processor node 34 also includes queue pairs (QP's) 64a and 64b which provide communication between process 60 and SANIC 42. Host processor node 34 also includes QP 64c which provides communication between process 62 and SANIC 42. A single SANIC, such as SANIC 42 in a host processor 34, can support thousands of QPs. By contrast, a SAN interface in an I/O adapter 35 typically supports less than ten QPs.

Each QP 64 includes a send work queue 66 and a receive work queue 68. A process, such as processes 60 and 62, calls an operating-system specific programming interface which is herein referred to as verbs, which place work items, referred to as work queue elements (WQEs) onto a QP 64. A WQE is executed by hardware in SANIC 42. SANIC 42 is coupled to SAN 32 via physical link 40. Send work queue 66 contains WQEs that describe data to be transmitted on the SAN 32 fabric. Receive work queue 68 contains WQEs that describe where to place incoming data from the SAN 32 fabric.

Host processor node 34 also includes completion queue 70a interfacing with process 60 and completion queue 70b interfacing with process 62. The completion queues 70 contain information about completed WQEs. The completion queues are employed to create a single point of completion notification for multiple QPs. A completion queue entry is a data structure on a completion queue 70 that describes a completed WQE. The completion queue entry contains sufficient information to determine the QP that holds the

completed WQE. A completion queue context is a block of information that contains pointers to, length, and other information needed to manage the individual completion queues.

Example WQEs include work items that initiate data communications employing channel semantics or memory semantics; work items that are instructions to hardware in SANIC 42 to set or alter remote memory access protections; and work items to delay the execution of subsequent WQEs posted in the same send work queue 66.

More specifically, example WQEs supported for send work queues 66 are as follows. A send buffer WQE is a channel semantic operation to push a local buffer to a remote QP's receive buffer. The send buffer WQE includes a gather list to combine several virtual contiguous local buffers into a single message that is pushed to a remote QP's receive buffer. The local buffer virtual addresses are in the address space of the process that created the local QP.

A remote direct memory access (RDMA) read WQE provides a memory semantic operation to read a virtually contiguous buffer on a remote node. The RDMA read WQE reads a virtually contiguous buffer on a remote endnode and writes the data to a scatter list of virtually contiguous local memory buffers. Similar to the send buffer WQE, the local buffer for the RDMA read WQE is in the address space of the process that created the local QP. The remote buffer is in the virtual address space of the process owning the remote QP targeted by the RDMA read WQE.

A RDMA write WQE provides a memory semantic operation to write a virtually contiguous buffer on a remote node. The RDMA write WQE contains a scatter list of locally virtually contiguous buffers and the virtual address of the remote buffer into which the local buffers are written.

A RDMA FetchOp WQE provides a memory semantic operation to perform an atomic operation on a remote word. The RDMA FetchOp WQE is a combined RDMA read, modify, and RDMA write operation. The RDMA FetchOp WQE can support several read-modify-write operations, such as Compare and Swap if equal.

A bind/unbind remote access key (RKey) WQE provides a command to SANIC hardware to modify the association of a RKey with a local virtually contiguous buffer. The RKey is part of each RDMA access and is used to validate that the remote process has permitted access to the buffer.

A delay WQE provides a command to SANIC hardware to delay processing of the QP's WQEs for a specific time interval. The delay WQE permits a process to meter the flow of operations into the SAN fabric.

In one embodiment, receive queues 68 only support one type of WQE, which is referred to as a receive buffer WQE. The receive buffer WQE provides a channel semantic operation describing a local buffer into which incoming send messages are written. The receive buffer WQE includes a scatter list describing several virtually contiguous local buffers. An incoming send message is written to these buffers. The buffer virtual addresses are in the address space of the process that created the local QP.

For IPC, a user-mode software process transfers data through QPs 64 directly from where the buffer resides in memory. In one embodiment, the transfer through the QPs bypasses the operating system and consumes few host instruction cycles. QPs 64 permit zero processor-copy data transfer with no operating system kernel involvement. The zero processor-copy data transfer provides for efficient support of high-bandwidth and low-latency communication.

Transport Services

When a QP 64 is created, the QP is set to provide a selected type of transport service. In one embodiment, a distributed computer system implementing the present invention supports four types of transport services.

A portion of a distributed computer system employing a reliable connection service to communicate between distributed processes is illustrated generally at 100 in Figure 3. Distributed computer system 100 includes a host processor node 102, a host processor node 104, and a host processor node 106. Host processor node 102 includes a process A indicated at 108. Host processor

node 104 includes a process B indicated at 110 and a process C indicated at 112. Host processor node 106 includes a process D indicated at 114.

Host processor node 102 includes a QP 116 having a send work queue 116a and a receive work queue 116b; a QP 118 having a send work queue 118a and receive work queue 118b; and a QP 120 having a send work queue 120a and a receive work queue 120b which facilitate communication to and from process A indicated at 108. Host processor node 104 includes a QP 122 having a send work queue 122a and receive work queue 122b for facilitating communication to and from process B indicated at 110. Host processor node 104 includes a QP 124 having a send work queue 124a and receive work queue 124b for facilitating communication to and from process C indicated at 112. Host processor node 106 includes a QP 126 having a send work queue 126a and receive work queue 126b for facilitating communication to and from process D indicated at 114.

The reliable connection service of distributed computer system 100 associates a local QP with one and only one remote QP. Thus, QP 116 is connected to QP 122 via a non-sharable resource connection 128 having a non-sharable resource connection 128a from send work queue 116a to receive work queue 122b and a non-sharable resource connection 128b from send work queue 122a to receive work queue 116b. QP 118 is connected to QP 124 via a non-sharable resource connection 130 having a non-sharable resource connection 130a from send work queue 118a to receive work queue 124b and a non-sharable resource connection 130b from send work queue 124a to receive work queue 118b. QP 120 is connected to QP 126 via a non-sharable resource connection 132 having a non-sharable resource connection 132a from send work queue 120a to receive work queue 126b and a non-sharable resource connection 132b from send work queue 126a to receive work queue 120b.

A send buffer WQE placed on one QP in a reliable connection service causes data to be written into the receive buffer of the connected QP. RDMA operations operate on the address space of the connected QP.

The reliable connection service requires a process to create a QP for each process which is to communicate with over the SAN fabric. Thus, if each of N

host processor nodes contain M processes, and all M processes on each node wish to communicate with all the processes on all the other nodes, each host processor node requires M^2 x (N – 1) QPs. Moreover, a process can connect a QP to another QP on the same SANIC.

In one embodiment, the reliable connection service is made reliable because hardware maintains sequence numbers and acknowledges all frame transfers. A combination of hardware and SAN driver software retries any failed communications. The process client of the QP obtains reliable communications even in the presence of bit errors, receive buffer underruns, and network congestion. If alternative paths exist in the SAN fabric, reliable communications can be maintained even in the presence of failures of fabric switches or links.

In one embodiment, acknowledgements are employed to deliver data reliably across the SAN fabric. In one embodiment, the acknowledgement is not a process level acknowledgment, because the acknowledgment does not validate the receiving process has consumed the data. Rather, the acknowledgment only indicates that the data has reached its destination.

A portion of a distributed computer system employing a reliable datagram service to communicate between distributed processes is illustrated generally at 150 in Figure 4. Distributed computer system 150 includes a host processor node 152, a host processor node 154, and a host processor node 156. Host processor node 152 includes a process A indicated at 158. Host processor node 154 includes a process B indicated at 160 and a process C indicated at 162. Host processor node 156 includes a process D indicated at 164.

Host processor node 152 includes QP 166 having send work queue 166a and receive work queue 166b for facilitating communication to and from process A indicated at 158. Host processor node 154 includes QP 168 having send work queue 168a and receive work queue 168b for facilitating communication from and to process B indicated at 160. Host processor node 154 includes QP 170 having send work queue 170a and receive work queue 170b for facilitating communication from and to process C indicated at 162. Host processor node 156 includes QP 172 having send work queue 172a and receive work queue

172b for facilitating communication from and to process D indicated at 164. In the reliable datagram service implemented in distributed computer system 150, the QPs are coupled in what is referred to as a connectionless transport service.

For example, a reliable datagram service 174 couples QP 166 to QPs 168, 170, and 172. Specifically, reliable datagram service 174 couples send work queue 166a to receive work queues 168b, 170b, and 172b. Reliable datagram service 174 also couples send work queues 168a, 170a, and 172a to receive work queue 166b.

The reliable datagram service permits a client process of one QP to communicate with any other QP on any other remote node. At a receive work queue, the reliable datagram service permits incoming messages from any send work queue on any other remote node.

In one embodiment, the reliable datagram service employs sequence numbers and acknowledgments associated with each message frame to ensure the same degree of reliability as the reliable connection service. End-to-end (EE) contexts maintain end-to-end specific state to keep track of sequence numbers, acknowledgments, and time-out values. The end-to-end state held in the EE contexts is shared by all the connectionless QPs communicating between a pair of endnodes. Each endnode requires at least one EE context for every endnode it wishes to communicate with in the reliable datagram service (e.g., a given endnode requires at least N EE contexts to be able to have reliable datagram service with N other endnodes).

The reliable datagram service greatly improves scalability because the reliable datagram service is connectionless. Therefore, an endnode with a fixed number of QPs can communicate with far more processes and endnodes with a reliable datagram service than with a reliable connection transport service. For example, if each of N host processor nodes contain M processes, and all M processes on each node wish to communicate with all the processes on all the other nodes, the reliable connection service requires $M^2 \times (N - 1)$ QPs on each node. By comparison, the connectionless reliable datagram service only requires

M QPs + (N –1) EE contexts on each node for exactly the same communications.

A third type of transport service for providing communications is a unreliable datagram service. Similar to the reliable datagram service, the unreliable datagram service is connectionless. The unreliable datagram service is employed by management applications to discover and integrate new switches, routers, and endnodes into a given distributed computer system. The unreliable datagram service does not provide the reliability guarantees of the reliable connection service and the reliable datagram service. The unreliable datagram service accordingly operates with less state information maintained at each endnode.

A fourth type of transport service is referred to as raw datagram service and is technically not a transport service. The raw datagram service permits a QP to send and to receive raw datagram frames. The raw datagram mode of operation of a QP is entirely controlled by software. The raw datagram mode of the QP is primarily intended to allow easy interfacing with traditional internet protocol, version 6 (IPv6) LAN-WAN networks, and further allows the SANIC to be used with full software protocol stacks to access transmission control protocol (TCP), user datagram protocol (UDP), and other standard communication protocols. Essentially, in the raw datagram service, SANIC hardware generates and consumes standard protocols layered on top of IPv6, such as TCP and UDP. The frame header can be mapped directly to and from an IPv6 header. Native IPv6 frames can be bridged into the SAN fabric and delivered directly to a QP to allow a client process to support any transport protocol running on top of IPv6. A client process can register with SANIC hardware in order to direct datagrams for a particular upper level protocol (e.g., TCP and UDP) to a particular QP. SANIC hardware can demultiplex incoming IPv6 streams of datagrams based on a next header field as well as the destination IP address.

SANIC and I/O Adapter Endnodes

An example host processor node is generally illustrated at 200 in Figure 5. Host processor node 200 includes a process A indicated at 202, a process B indicated at 204, and a process C indicated at 206. Host processor 200 includes a SANIC 208 and a SANIC 210. As discussed above, a host processor endnode or an I/O adapter endnode can have one or more SANICs. SANIC 208 includes a SAN link level engine (LLE) 216 for communicating with SAN fabric 224 via link 217 and an LLE 218 for communicating with SAN fabric 224 via link 219. SANIC 210 includes an LLE 220 for communicating with SAN fabric 224 via link 221 and an LLE 222 for communicating with SAN fabric 224 via link 223. SANIC 208 communicates with process A indicated at 202 via QPs 212a and 212b. SANIC 208 communicates with process B indicated at 204 via QPs 212c-212n. Thus, SANIC 208 includes N QPs for communicating with processes A and B. SANIC 210 includes QPs 214a and 214b for communicating with process B indicated at 204. SANIC 210 includes QPs 214c-214n for communicating with process C indicated at 206. Thus, SANIC 210 includes N QPs for communicating with processes B and C.

An LLE runs link level protocols to couple a given SANIC to the SAN fabric. RDMA traffic generated by a SANIC can simultaneously employ multiple LLEs within the SANIC which permits striping across LLEs. Striping refers to the dynamic sending of frames within a single message to an endnode's QP through multiple fabric paths. Striping across LLEs increases the bandwidth for a single QP as well as provides multiple fault tolerant paths. Striping also decreases the latency for message transfers. In one embodiment, multiple LLEs in a SANIC are not visible to the client process generating message requests. When a host processor includes multiple SANICs, the client process must explicitly move data on the two SANICs in order to gain parallelism. A single QP cannot be shared by SANICS. Instead a QP is owned by one local SANIC.

The following is an example naming scheme for naming and identifying endnodes in one embodiment of a distributed computer system according to the present invention. A host name provides a logical identification for a host node, such as a host processor node or I/O adapter node. The host name identifies the

endpoint for messages such that messages are destine for processes residing on an endnode specified by the host name. Thus, there is one host name per node, but a node can have multiple SANICs.

A globally unique ID (GUID) identifies a transport endpoint. A transport endpoint is the device supporting the transport QPs. There is one GUID associated with each SANIC.

A local ID refers to a short address ID used to identify a SANIC within a single subnet. In one example embodiment, a subnet has up 2^{16} endnodes, switches, and routers, and the local ID (LID) is accordingly 16 bits. A source LID (SLID) and a destination LID (DLID) are the source and destination LIDs used in a local network header. A LLE has a single LID associated with the LLE, and the LID is only unique within a given subnet. One or more LIDs can be associated with each SANIC.

An internet protocol (IP) address (e.g., a 128 bit IPv6 ID) addresses a SANIC. The SANIC, however, can have one or more IP addresses associated with the SANIC. The IP address is used in the global network header when routing frames outside of a given subnet. LIDs and IP addresses are network endpoints and are the target of frames routed through the SAN fabric. All IP addresses (e.g., IPv6 addresses) within a subnet share a common set of high order address bits.

In one embodiment, the LLE is not named and is not architecturally visible to a client process. In this embodiment, management software refers to LLEs as an enumerated subset of the SANIC.

Switches and Routers

A portion of a distributed computer system is generally illustrated at 250 in Figure 6. Distributed computer system 250 includes a subnet A indicated at 252 and a subnet B indicated at 254. Subnet A indicated at 252 includes a host processor node 256 and a host processor node 258. Subnet B indicated at 254 includes a host processor node 260 and host processor node 262. Subnet A indicated at 252 includes switches 264a-264c. Subnet B indicated at 254

includes switches 266a-266c. Each subnet within distributed computer system 250 is connected to other subnets with routers. For example, subnet A indicated at 252 includes routers 268a and 268b which are coupled to routers 270a and 270b of subnet B indicated at 254. In one example embodiment, a subnet has up to 2^{16} endnodes, switches, and routers.

A subnet is defined as a group of endnodes and cascaded switches that is managed as a single unit. Typically, a subnet occupies a single geographic or functional area. For example, a single computer system in one room could be defined as a subnet. In one embodiment, the switches in a subnet can perform very fast worm-hole or cut-through routing for messages.

A switch within a subnet examines the DLID that is unique within the subnet to permit the switch to quickly and efficiently route incoming message frames. In one embodiment, the switch is a relatively simple circuit, and is typically implemented as a single integrated circuit. A subnet can have hundreds to thousands of endnodes formed by cascaded switches.

As illustrated in Figure 6, for expansion to much larger systems, subnets are connected with routers, such as routers 268 and 270. The router interprets the IP destination ID (e.g., IPv6 destination ID) and routes the IP like frame.

In one embodiment, switches and routers degrade when links are over utilized. In this embodiment, link level back pressure is used to temporarily slow the flow of data when multiple input frames compete for a common output. However, link or buffer contention does not cause loss of data. In one embodiment, switches, routers, and endnodes employ a link protocol to transfer data. In one embodiment, the link protocol supports an automatic error retry. In this example embodiment, link level acknowledgments detect errors and force retransmission of any data impacted by bit errors. Link-level error recovery greatly reduces the number of data errors that are handled by the end-to-end protocols. In one embodiment, the user client process is not involved with error recovery no matter if the error is detected and corrected by the link level protocol or the end-to-end protocol.

An example embodiment of a switch is generally illustrated at 280 in Figure 7. Each I/O path on a switch or router has an LLE. For example, switch 280 includes LLEs 282a-282h for communicating respectively with links 284a-284h.

The naming scheme for switches and routers is similar to the above-described naming scheme for endnodes. The following is an example switch and router naming scheme for identifying switches and routers in the SAN fabric. A switch name identifies each switch or group of switches packaged and managed together. Thus, there is a single switch name for each switch or group of switches packaged and managed together.

Each switch or router element has a single unique GUID. Each switch has one or more LIDs and IP addresses (e.g., IPv6 addresses) that are used as an endnode for management frames.

Each LLE is not given an explicit external name in the switch or router. Since links are point-to-point, the other end of the link does not need to address the LLE.

Virtual Lanes

Switches and routers employ multiple virtual lanes within a single physical link. As illustrated in Figure 6, physical links 272 connect endnodes, switches, and routers within a subnet. WAN or LAN connections 274 typically couple routers between subnets. Frames injected into the SAN fabric follow a particular virtual lane from the frame's source to the frame's destination. At any one time, only one virtual lane makes progress on a given physical link. Virtual lanes provide a technique for applying link level flow control to one virtual lane without affecting the other virtual lanes. When a frame on one virtual lane blocks due to contention, quality of service (QoS), or other considerations, a frame on a different virtual lane is allowed to make progress.

Virtual lanes are employed for numerous reasons, some of which are as follows. Virtual lanes provide QoS. In one example embodiment, certain virtual lanes are reserved for high priority or isonchronous traffic to provide QoS.

Virtual lanes provide deadlock avoidance. Virtual lanes allow topologies that contain loops to send frames across all physical links and still be assured the loops won't cause back pressure dependencies that might result in deadlock.

Virtual lanes alleviate head-of-line blocking. With virtual lanes, a blocked frames can pass a temporarily stalled frame that is destined for a different final destination.

In one embodiment, each switch includes its own crossbar switch. In this embodiment, a switch propagates data from only one frame at a time, per virtual lane through its crossbar switch. In another words, on any one virtual lane, a switch propagates a single frame from start to finish. Thus, in this embodiment, frames are not multiplexed together on a single virtual lane.

Paths in SAN fabric

Referring to Figure 6, within a subnet, such as subnet A indicated at 252 or subnet B indicated at 254, a path from a source port to a destination port is determined by the LID of the destination SANIC port. Between subnets, a path is determined by the IP address (e.g., IPv6 address) of the destination SANIC port.

In one embodiment, the paths used by the request frame and the request frame's corresponding positive acknowledgment (ACK) or negative acknowledgment (NAK) frame are not required to be symmetric. In one embodiment employing oblivious routing, switches select an output port based on the DLID. In one embodiment, a switch uses one set of routing decision criteria for all its input ports. In one example embodiment, the routing decision criteria is contained in one routing table. In an alternative embodiment, a switch employs a separate set of criteria for each input port.

Each port on an endnode can have multiple IP addresses. Multiple IP addresses can be used for several reasons, some of which are provided by the following examples. In one embodiment, different IP addresses identify different partitions or services on an endnode. In one embodiment, different IP

addresses are used to specify different QoS attributes. In one embodiment, different IP addresses identify different paths through intra-subnet routes.

In one embodiment, each port on an endnode can have multiple LIDs. Multiple LIDs can be used for several reasons some of which are provided by the following examples. In one embodiment, different LIDs identify different partitions or services on an endnode. In one embodiment, different LIDs are used to specify different QoS attributes. In one embodiment, different LIDs specify different paths through the subnet.

A one-to-one correspondence does not necessarily exist between LIDs and IP addresses, because a SANIC can have more or less LIDs than IP addresses for each port. For SANICs with redundant ports and redundant conductivity to multiple SAN fabrics, SANICs can, but are not required to, use the same LID and IP address on each of its ports.

Data Transactions

Referring to Figure 1, a data transaction in distributed computer system 30 is typically composed of several hardware and software steps. A client process of a data transport service can be a user-mode or a kernel-mode process. The client process accessess SANIC 42 hardware through one or more QPs, such as QPs 64 illustrated in Figure 2. The client process calls an operating-system specific programming interface which is herein referred to as verbs. The software code implementing the verbs intern posts a WQE to the given QP work queue.

There are many possible methods of posting a WQE and there are many possible WQE formats, which allow for various cost/performance design points, but which do not affect interoperability. A user process, however, must communicate to verbs in a well-defined manner, and the format and protocols of data transmitted across the SAN fabric must be sufficiently specified to allow devices to interoperate in a heterogeneous vendor environment.

In one embodiment, SANIC hardware detects WQE posting and accesses the WQE. In this embodiment, the SANIC hardware translates and validates the

WQEs virtual addresses and accesses the data. In one embodiment, an outgoing message buffer is split into one or more frames. In one embodiment, the SANIC hardware adds a transport header and a network header to each frame. The transport header includes sequence numbers and other transport information. The network header includes the destination IP address or the DLID or other suitable destination address information. The appropriate local or global network header is added to a given frame depending on if the destination endnode resides on the local subnet or on a remote subnet.

A frame is a unit of information that is routed through the SAN fabric. The frame is an endnode-to-endnode construct, and is thus created and consumed by endnodes. Switches and routers neither generate nor consume request frames or acknowledgment frames. Instead switches and routers simply move request frames or acknowledgment frames closer to the ultimate destination. Routers, however, modify the frame's network header when the frame crosses a subnet boundary. In traversing a subnet, a single frame stays on a single virtual lane.

When a frame is placed onto a link, the frame is further broken down into flits. A flit is herein defined to be a unit of link-level flow control and is a unit of transfer employed only on a point-to-point link. The flow of flits is subject to the link-level protocol which can perform flow control or retransmission after an error. Thus, flit is a link-level construct that is created at each endnode, switch, or router output port and consumed at each input port. In one embodiment, a flit contains a header with virtual lane error checking information, size information, and reverse channel credit information.

If a reliable transport service is employed, after a request frame reaches its destination endnode, the destination endnode sends an acknowledgment frame back to the sender endnode. The acknowledgment frame permits the requestor to validate that the request frame reached the destination endnode. An acknowledgment frame is sent back to the requestor after each request frame. The requestor can have multiple outstanding requests before it receives any

acknowledgments. In one embodiment, the number of multiple outstanding requests is determined when a QP is created.

Example Request and Acknowledgment Transactions

Figures 8, 9A, 9B, 10A, and 10B together illustrate example request and acknowledgment transactions. In Figure 8, a portion of a distributed computer system is generally illustrated at 300. Distributed computer system 300 includes a host processor node 302 and a host processor node 304. Host processor node 302 includes a SANIC 306. Host processor node 304 includes a SANIC 308. Distributed computer system 300 includes a SAN fabric 309 which includes a switch 310 and a switch 312. SAN fabric 309 includes a link 314 coupling SANIC 306 to switch 310; a link 316 coupling switch 310 to switch 312; and a link 318 coupling SANIC 308 to switch 312.

In the example transactions, host processor node 302 includes a client process A indicated at 320. Host processor node 304 includes a client process B indicated at 322. Client process 320 interacts with SANIC hardware 306 through QP 324. Client process 322 interacts with SANIC hardware 308 through QP 326. QP 324 and 326 are software data structures. QP 324 includes send work queue 324a and receive work queue 324b. QP 326 includes send work queue 326a and receive work queue 326b.

Process 320 initiates a message request by posting WQEs to send queue 324a. Such a WQE is illustrated at 330 in Figure 9A. The message request of client process 320 is referenced by a gather list 332 contained in send WQE 330. Each entry in gather list 332 points to a virtually contiguous buffer in the local memory space containing a part of the message, such as indicated by virtual contiguous buffers 334a-334d, which respectively hold message 0, parts 0, 1, 2, and 3.

Referring to Figure 9B, hardware in SANIC 306 reads WQE 330 and packetizes the message stored in virtual contiguous buffers 334a-334d into frames and flits. As illustrated in Figure 9B, all of message 0, part 0 and a portion of message 0, part 1 are packetized into frame 0, indicated at 336a. The

rest of message 0, part 1 and all of message 0, part 2, and all of message 0, part 3 are packetized into frame 1, indicated at 336b. Frame 0 indicated at 336a includes network header 338a and transport header 340a. Frame 1 indicated at 336b includes network header 338b and transport header 340b.

As indicated in Figure 9B, frame 0 indicated at 336a is partitioned into flits 0-3, indicated respectively at 342a-342d. Frame 1 indicated at 336b is partitioned into flits 4-7 indicated respectively at 342e - 342h. Flits 342a through 342h respectively include flit headers 344a-344h.

Frames are routed through the SAN fabric, and for reliable transfer services, are acknowledged by the final destination endnode. If not successively acknowledged, the frame is retransmitted by the source endnode. Frames are generated by source endnodes and consumed by destination endnodes. The switches and routers in the SAN fabric neither generate nor consume frames.

Flits are the smallest unit of flow control in the network. Flits are generated and consumed at each end of a physical link. Flits are acknowledged at the receiving end of each link and are retransmitted in response to an error.

Referring to Figure 10A, the send request message 0 is transmitted from SANIC 306 in host processor node 302 to SANIC 308 in host processor node 304 as frames 0 indicated at 336a and frame 1 indicated at 336b. ACK frames 346a and 346b, corresponding respectively to request frames 336a and 336b, are transmitted from SANIC 308 in host processor node 304 to SANIC 306 in host processor node 302.

In Figure 10A, message 0 is being transmitted with a reliable transport service. Each request frame is individually acknowledged by the destination endnode (e.g., SANIC 308 in host processor node 304).

Figure 10B illustrates the flits associated with the request frames 336 and acknowledgment frames 346 illustrated in Figure 10A passing between the host processor endnodes 302 and 304 and the switches 310 and 312. As illustrated in Figure 10B, an ACK frame fits inside one flit. In one embodiment, one acknowledgment flit acknowledges several flits.

As illustrated in Figure 10B, flits 342a-h are transmitted from SANIC 306 to switch 310. Switch 310 consumes flits 342a-h at its input port, creates flits 348a-h at its output port corresponding to flits 342a-h, and transmits flits 348a-h to switch 312. Switch 312 consumes flits 348a-h at its input port, creates flits 350a-h at its output port corresponding to flits 348a-h, and transmits flits 350a-h to SANIC 308. SANIC 308 consumes flits 350a-h at its input port. An acknowledgment flit is transmitted from switch 310 to SANIC 306 to acknowledge the receipt of flits 342a-h. An acknowledgment flit 354 is transmitted from switch 312 to switch 310 to acknowledge the receipt of flits 348a-h. An acknowledgment flit 356 is transmitted from SANIC 308 to switch 312 to acknowledge the receipt of flits 350a-h.

Acknowledgment frame 346a fits inside of flit 358 which is transmitted from SANIC 308 to switch 312. Switch 312 consumes flits 358 at its input port, creates flit 360 corresponding to flit 358 at its output port, and transmits flit 360 to switch 310. Switch 310 consumes flit 360 at its input port, creates flit 362 corresponding to flit 360 at its output port, and transmits flit 362 to SANIC 306. SANIC 306 consumes flit 362 at its input port. Similarly, SANIC 308 transmits acknowledgment frame 346b in flit 364 to switch 312. Switch 312 creates flit 366 corresponding to flit 364, and transmits flit 366 to switch 310. Switch 310 creates flit 368 corresponding to flit 366, and transmits flit 368 to SANIC 306.

Switch 312 acknowledges the receipt of flits 358 and 364 with acknowledgment flit 370, which is transmitted from switch 312 to SANIC 308. Switch 310 acknowledges the receipt of flits 360 and 366 with acknowledgment flit 372, which is transmitted to switch 312. SANIC 306 acknowledges the receipt of flits 362 and 368 with acknowledgment flit 374 which is transmitted to switch 310.

Architecture Layers and Implementation Overview

A host processor endnode and an I/O adapter endnode typically have quite different capabilities. For example, an example host processor endnode might support four ports, hundreds to thousands of QPs, and allow incoming

RDMA operations, while an attached I/O adapter endnode might only support one or two ports, tens of QPs, and not allow incoming RDMA operations. A low-end attached I/O adapter alternatively can employ software to handle much of the network and transport layer functionality which is performed in hardware (e.g., by SANIC hardware) at the host processor endnode.

One embodiment of a layered architecture for implementing the present invention is generally illustrated at 400 in diagram form in Figure 11. The layered architecture diagram of Figure 11 shows the various layers of data communication paths, and organization of data and control information passed between layers.

Host SANIC endnode layers are generally indicated at 402. The host SANIC endnode layers 402 include an upper layer protocol 404; a transport layer 406; a network layer 408; a link layer 410; and a physical layer 412.

Switch or router layers are generally indicated at 414. Switch or router layers 414 include a network layer 416; a link layer 418; and a physical layer 420.

I/O adapter endnode layers are generally indicated at 422. I/O adapter endnode layers 422 include an upper layer protocol 424; a transport layer 426; a network layer 428; a link layer 430; and a physical layer 432.

The layered architecture 400 generally follows an outline of a classical communication stack. The upper layer protocols employ verbs to create messages at the transport layers. The transport layers pass messages to the network layers. The network layers pass frames down to the link layers. The link layers pass flits through physical layers. The physical layers send bits or groups of bits to other physical layers. Similarly, the link layers pass flits to other link layers, and don't have visibility to how the physical layer bit transmission is actually accomplished. The network layers only handle frame routing, without visibility to segmentation and reassembly of frames into flits or transmission between link layers.

Bits or groups of bits are passed between physical layers via links 434. Links 434 can be implemented with printed circuit copper traces, copper cable, optical cable, or with other suitable links.

The upper layer protocol layers are applications or processes which employ the other layers for communicating between endnodes.

The transport layers provide end-to-end message movement. In one embodiment, the transport layers provide four types of transport services as described above which are reliable connection service; reliable datagram service; unreliable datagram service; and raw datagram service.

The network layers perform frame routing through a subnet or multiple subnets to destination endnodes.

The link layers perform flow-controlled, error controlled, and prioritized frame delivery across links.

The physical layers perform technology-dependent bit transmission and reassembly into flits.

Memory Management

A computer system is illustrated generally at 500 in Figure 12. Computer system 500 includes at least one processor, such as processor 502, for performing sequences of logical operations. Computer system 500 also includes memory 504 for storing instructions and data for use by processor 502. An operating system 506 is stored in memory 504 and controls processor 502 and memory 504 for system operations and for executing processes, such as user process 508 stored in memory 504. Memory 504 typically includes random access memory (RAM), non-volatile memory, and a hard disk drive, but can include any known type of memory storage. Memory 504 can be located local to processor 502 or remote from processor 502. In addition, memory 504 can be shared among numerous other processors.

A consumer process, which can be a kernel process or a user process, such as user process 508, operates under an address space which is pointed to by an effective address of the consumer process. For example, user process 508

operates under effective address space 510. For most advanced processors, the effective address is typically a pointer into a data structure that contains a virtual address. The data structure containing the virtual address is typically referred to as a segment table, such as segment table 512 containing a segment table entry 514 holding the virtual address. The effective address pointer is typically controlled by an operating system kernel process, such as virtual memory manager (VMM) 518 of kernel process 516. A user process cannot access a virtual address unless the operating system has allowed the user process to access the virtual address. The operating system allows the access of a virtual address by creating a pointer to the virtual address, including the pointer in the segment table, and allowing the user process to employ the pointer as part of an effective address. The segment table is typically unique to a specific process.

An example effective address 530 is generally illustrated in Figure 13. Example effective address 530 contains a pointer to a segment table entry 532 (i.e., effective segment ID); a memory page number 534; and a page offset 536 into the memory page. The pointer to the segment table entry is used to extract the segment table entry from the segment table.

An example segment table entry 540 is illustrated generally in Figure 14. Example segment table entry 540 includes an effective segment ID 532' corresponding to pointer 532 of effective address 530; the upper bits of the virtual address 542 that the effective segment ID is mapped to; and other fields 544, such as segment valid bit field 546.

A physical address is the address actually employed to address the physical memory. A virtual address provides a much wider address space than the actual physical address space. In one embodiment, a user process cannot directly access the virtual address space.

An example virtual address 550 is generally illustrated in Figure 15. Example virtual address 550 includes the upper bits of the virtual address 542' obtained from the segment table entry 540 that maps the effective address to the virtual address; memory page number 534' providing a pointer to the memory page; and page offset 536' providing an offset into the memory page. A physical

address is generated by employing the upper virtual address 542' and page number 534' to extract a page table entry from a page table, such as page table entry 522 of page table 520 stored in memory 504 of Figure 1.

An example page table entry 552 is illustrated generally in Figure 16. Example page table entry 552 includes the upper virtual address 542"; physical memory page number 534"; and several other fields 554, such as page valid field 556, and page protection field 558.

In one embodiment of a distributed computer system according to the present invention, such as distributed computer system 30 of Figure 1, one memory management technique is employed for local memory access protection and a separate, but related, memory management technique is employed for remote memory access protection.

When computer system 500 is part of a distributed computer system, such as distributed computer system 30 of Figure 1, computer system 500 includes a SANIC 524 for sourcing and sinking frames transmitted on SAN fabric 528. A SANIC driver process (SANIC driver) 526 is stored in memory 504 and controls the SANIC hardware in SANIC 524 and performs other SAN fabric interface functions.

The local memory access protection is tied directly to the operating system kernel and hardware platform's VVM process, such as VMM 518. Local memory access protection assures that a consumer process only accesses the effective address regions authorized by the local operating system kernel process, such as kernel process 516. For example, in a distributed computer system, a consumer process residing on one of the host processor nodes cannot use a SANIC driver, such as SANIC driver 526, to gain unauthorized access to any virtual or physical address region.

The remote memory access protection is under the control of a consumer, which can be a kernel process or a user process. The remote memory access protection assures a process executing on a remote node can only access the effective memory regions authorized by the local consumer process which is

authorized through the local memory access protection by the local operating system VMM.

Local Memory Protection

Referring to Figure 12, local memory access protection is performed by executing SANIC driver 526 in a privileged mode. SANIC driver 526 is also tied to the processor platform of computer system 500. In addition, VMM 518 of operating system kernel process 516 preferably controls local memory access protection instead of having local memory access protection being directly controlled by SANIC driver 526. Local memory access protection is preferably controlled by VMM 518 for several reasons including the following reasons.

The VMM controlling the local memory access protection allows the segment table 512 contents to be obtained for the calling process. In order to extract the virtual address from the effective address used by the calling process, SANIC driver 526 needs at least a copy to the segment page table. In one embodiment, SANIC driver 526 requires a pointer to the actual segment table 512 employed by VMM 518.

The VMM 518 controlling local memory access protection allows the segment table entries 514 to be maintained current if the segment table entries change. For example, segment table entries 514 can changed when a process extends the process's memory region. In order to maintain the segment table entries 514 current, the VMM 518 needs to keep SANIC driver 526 informed of all segment table 512 changes, such as when new segment table entries 514 are added to segment table 512. All segment table entries 514 need to contain similar attributes as the operating system VMM 518's segment table, such as an entry valid attribute. In addition, SANIC driver 526 needs a method of obtaining the latest segment table entries 514 in case SANIC 524 caches some of the latest segment table entries.

The VMM controlling local memory access protection also allows page table entries 522 of page table 520 to be maintained current. In order to maintain page table entries 522 current, VMM 518 keeps SANIC driver 526 informed of

all page table 520 changes, such as page swaps in and page swaps out. All page table entries 522 contain similar attributes as the operating system VMM 518's page table, such as entry valid and protection level attributes. In one embodiment, VMM 518 can grant SANIC 524 access to page table 520.

SANIC driver 526 can employ similar mechanisms as employed by processor 502 to restricted access by a calling process to only the effective address the process is authorization to access. One embodiment of a suitable mechanism for managing local memory access protection is illustrated generally at 600 in Figure 17. At step 602, an OpenQP verb is executed to open a QP, such as QP 529, to permit a calling process to communicate with a similar process on a remote node. QP 529 includes a send work queue and a receive work queue. When QP 529 is created, SANIC driver 526 and SANIC 524 save a set of state information about the specific QP instance, such as QP state 531. Example information stored in QP state 531 includes the process using the QP; the size of the QP; the type of transfers the calling process desires for the QP; and the like. The information in QP state 531 is created by SANIC driver 526 and is not accessible by the calling process.

At step 604, when QP 529 is opened, SANIC driver 526 issues a call to the local operating system VMM 518 and obtains the segment table 512 for the calling process that issued the OpenQP verb. This call from SANIC driver 526 to VMM 518 obtains the contents of segment table 512 for the calling process and also registers SANIC driver 526 with the local operating system VMM 518. By registering SANIC driver 526 with VMM 518, VMM 518 is able to notify SANIC driver 526 whenever segment table 512 is updated for the calling process. The segment table 512 is then stored as part of QP state 531. For a process that has many QPs open, the common state information can be stored only once for all the QPs the given process has open. In addition, the QP state can be a pointer to the actual operating system segment table.

At step 606, whenever the process passes an effective address to SANIC driver 526, the segment table 512 stored during QP 529 creation time is employed to translate the effective address into a virtual address.

At step 608, the local operating system VMM 518 informs SANIC driver 526 of any update in segment table 512 by executing a SegmentTableUpdate verb. The SegmentTableUpdate verb is executed by VMM 518 whenever an entry is a segment table 512 is added, deleted, or changed.

At step 610, SANIC driver 526 updates QP state 531 to reflect the change in the QP context(s) associated with the updated segment table.

In one embodiment, SANIC 524 employs the same segment table as VMM 518. In this embodiment, updates to the segment table employed by VMM 518 are by definition known by SANIC 524 without execution of the SegmentTableUpdate verb.

At step 612, SANIC 524 maps the virtual address into a physical address by executing a MapLocalPage verb. Even though steps 602 – 608 permit SANIC 524 to map an effective address into a physical address, SANIC 524 must also be able to map a virtual address into a physical address, because many operating system kernel programs use virtual addresses instead of effective addresses for I/O operations.

For example, a disk device driver kernel process typically obtains a virtual address instead of an effective address path to the disk device driver by a process (e.g., file system) that called the disk device driver. In one embodiment, the operating system kernel processes, such as a disk device driver, can use a kernel call to translate the virtual address to a physical address and then pass the physical address to SANIC 524. Alternatively, the operating system kernel processes, such as the disk device driver, can pass the virtual address to SANIC 524 and allow SANIC 524 to translate the virtual address to the physical address by employing the same page table as used by the operating system VMM 518.

At step 614, the consumer can then pass an opaque address to the remote node or alternatively pass the actual physical address to the remote node. In either case, the address is protected by a remote key (RKey) as discussed in detail below. The SANIC would make the remote address opaque versus the actual physical address so as to not expose the physical address to a process in the remote node that will use the physical address for remote direct memory

access (RDMA) operations. Exposing the physical address to the remote node, can expose the physical address on any node in the distributed computer system including the local node, because the remote node could use IPC to pass back the physical address to the local node.

In one embodiment, SANIC 524 maintains a private page table, which is not accessible by the calling process. In another embodiment, SANIC 524 has access to the operating system VMM's page table. In the embodiment where SANIC 524 maintains its own private page table, each SANIC needs to have a page table. Each SANIC maintaining its own page table is not very efficient when multiple SANICs are used per node. In the embodiment where SANIC 524 has access to the operating system page table, SANIC driver 526 needs access to operating system VMM 518 system calls that are used to swap pages in the page table to allow unpinned DMAs from remote nodes. The local node passes a remote node a RKey and memory region pair as described below, but without pinning the memory region. The remote node later attempts a RDMA to the memory region, but the page is swapped out, and at this point SANIC 524 needs to swap in the missing page table entry 522.

The above-described mechanism 600 for managing local memory access protection does not require any form of explicit process identification, because SANIC driver 526 is effectively employing the same memory protection mechanism as the local operating system VMM 518. One problem with the above-mechanism for managing local memory access protection is that the page table can become as big as the local operating system's VMM, which can be alleviated to some degree by implementing an inverted page table.

Remote Memory Protection

A distributed computer system having one embodiment of a mechanism for managing remote memory access protection according to the present invention is illustrated generally at 700 in Figure 18. Distributed computer system 700 includes a host processor node 702 and a host processor node 703 which both contain mechanisms similar to the mechanism 600 illustrated in

Figure 17 to manage local memory access protection, and both contain components which operate similar to the components described above for computer system 500 of Figure 12.

A host processor node 702 includes at least one processor such as processor 704, for performing sequences of logical operations, and a memory 706 for storing instructions and data for use by processor 704. An operating system 708 containing a kernel process 710 is stored in memory 706 and controls processor 704 and memory 706 for system operations and for executing processes. Operating system kernel process 710 includes a VMM 712. Host processor node 702 also includes SANIC 714 for sourcing and sinking frames transmitted on a SAN fabric 722. A SANIC driver 716 stored in memory 706 controls hardware in SANIC 714 and performs other SAN fabric interface functions. Host processor node 702 includes a user process 718 operating under an effective address space 720.

Host processor node 703 includes at least one processor such as processor 724, for performing sequences of logical operations, and a memory 726 for storing instructions and data for use by processor 724. An operating system 728 containing a kernel process 730 is stored in memory 726 and controls processor 724 and memory 726 for system operations and for executing processes. Operating system kernel process 730 includes a VMM 732. Host processor node 703 also includes SANIC 734 for sourcing and sinking frames transmitted on a SAN fabric 722. A SANIC driver 716 stored in memory 706 controls hardware and SANIC 714 and performs other SAN fabric interface functions. Host processor node 703 includes a user process 738 operating under an effective address space 740.

In an example remote key (RKey) binding operation, a consumer user process, such as user process 718 stored in memory 706 of host processor node 702, or a consumer kernel process (e.g., device driver) can function as a calling consumer process for accessing memory space in memory 706 of host processor node 702, such as effective address space 720 of memory 706.

The RKey binding operation is part of an RKey memory protection mechanism. The RKey binding operation binds a key to a contiguous memory address range, which can be addressed by an effective or virtual address. The contiguous memory address range can have an arbitrary size and an arbitrary alignment. In one embodiment, the RKey memory protection mechanism according to the present invention provides byte level granularity memory access protection.

A user calling process, such as user process 718, can only bind a RKey to an effective address pointing to an address space in memory accessible by the user calling process, such as effective address space 720. On the other hand, a kernel calling process can bind a RKey to either an effective address or a virtual address. Binding a RKey to a virtual address is useful for a kernel calling process when the kernel calling process has received an I/O request which only contains a virtual address plus length from another kernel process. For example, binding a RKey to a virtual address is useful for most kernel device drivers.

In an example operation, a kernel process, such as a file system, issues an I/O request to the kernel device driver. The I/O request from the file system kernel process contains a virtual address plus length, and the type of operation to be performed, such as read operation, disk address operation, or other type of operation. The kernel device driver needs to bind the virtual address plus length to a RKey before employing the virtual address for data transfers to/from the file system kernel process.

In an example RKey binding operation, user process 718 of host processor node 702 has access to an effective address 744 pointing to effective address space 720. User process 718 employs a BindRKey verb to associate effective address 744 to a RKey 746. Host processor node 702 creates a message 748 containing effective address 744 and RKey 746 and transmits message 748 to remote host processor node 703. Later, user process 738 of host processor node 703 employs effective address 744 and RKey 746 to perform a remote DMA (RDMA) operation to access effective address space 720 in host processor node 702. To perform the RDMA operation, host processor node 703

creates a message 750 containing effective address 744 and RKey 746 which is transmitted over SAN fabric 722 to host processor node 702.

One example embodiment of an RKey protection memory mechanism according to the present invention employs four verbs which are AllocateRKey verb; DeallocateRKey verb; BindRKey verb; and UnbindRKey verb.

A consumer process employs the AllocateRKey verb to obtain one or more RKeys. The consumer process employs the BindRKey verb to bind a RKey to a contiguous memory address range. The contiguous memory address range can have an arbitrary size and arbitrary alignment (i.e., location). The contiguous memory address range can be specified by an effective address or a virtual address. However, a consumer user calling process can only use an effective address. A consumer kernel calling process can use an effective address or a virtual address to specify the contiguous memory address range.

If the consumer process is authorized to access the contiguous memory address range to which the RKey will be bound, the BindRKey verb completes successively. If the consumer process is not authorized to access the contiguous memory address range, an invalid parameter result is returned.

The BindRKey verb does not require operating system kernel process involvement. Thus, a light-weight efficient remote memory access protection mechanism embodiment employs hardware to directly execute the BindRKey verb and to not involve operating system kernel software.

In one embodiment, a consumer process employs the BindRKey verb with a null value for the address and a zero value for the length of the contiguous memory address range to implement the UnbindRKey verb to unbind a RKey from a contiguous memory address range. If the consumer process is authorized to access the contiguous memory address range to which the RKey is bound, the UnbindRKey verb completes successfully. If the consumer process is not authorized to access the contiguous memory address range to which the RKey is bound, an invalid parameter result is returned.

In one embodiment, a consumer process can reuse RKeys, but a single RKey cannot be used to protect more than a single memory region at a given instant.

RKeys can be used by all QPs associated with the same address space bound to the RKey. For example, if a process has ten QPs associated to it, each QP can support RDMA access with the same RKey.

The RKey can be used by any number of remote nodes. For example, if a process has a single connecting QP, then only the connected process on the one remote node can use the RKey. However, if a process has many connected QPs, the RKey can be used by all the processes having access to the connected QP. A process with a datagram node QP (e.g., a QP supporting either reliable datagram or unreliable datagram) that allows RDMA operations, can accept RDMA operations from any node in the distributed computer system that obtains the RKey.

In one embodiment, a consumer process employs the DeallocateRKey verb to retire a remote key that was previously obtained through an AllocateRKey verb.

If a RKey and corresponding contiguous memory address range bound to the RKey supplied by a remote node do not match the bound RKey and contiguous memory address range in the local node, the remote node is not granted access to the contiguous memory address range. If the contiguous memory address range is invalid (e.g., no segment table entry for the contiguous memory address range), the remote node is not granted access to the contiguous memory address range. If a virtual address was not previously mapped by the operating system through a map local page verb, the SANIC's default error handler process receives an asynchronous error result to handle the resulting page fault condition.

In one embodiment, an RKey is a one-shot validating mechanism such that as soon as the RKey is used for a RDMA operation, the translation for the RKey is disabled.

Although specific embodiments have been illustrated and described herein for purposes of description of the preferred embodiment, it will be appreciated by those of ordinary skill in the art that a wide variety of alternate and/or equivalent implementations calculated to achieve the same purposes may be substituted for the specific embodiments shown and described without departing from the scope of the present invention. Those with skill in the chemical, mechanical, electro-mechanical, electrical, and computer arts will readily appreciate that the present invention may be implemented in a very wide variety of embodiments. This application is intended to cover any adaptations or variations of the preferred embodiments discussed herein. Therefore, it is manifestly intended that this invention be limited only by the claims and the equivalents thereof.

WHAT IS CLAIMED IS:

1. A method of managing memory in a distributed computer system, the method comprising:

binding a remote key to a first address representing a contiguous memory address range at a first endnode;

sending the bound remote key and first address from the first endnode to a second endnode on a communication fabric; and

performing a remote direct memory access operation from the second endnode to access the contiguous memory address range including sending the bound remote key and the first address from the second endnode to the first endnode on the communication fabric.

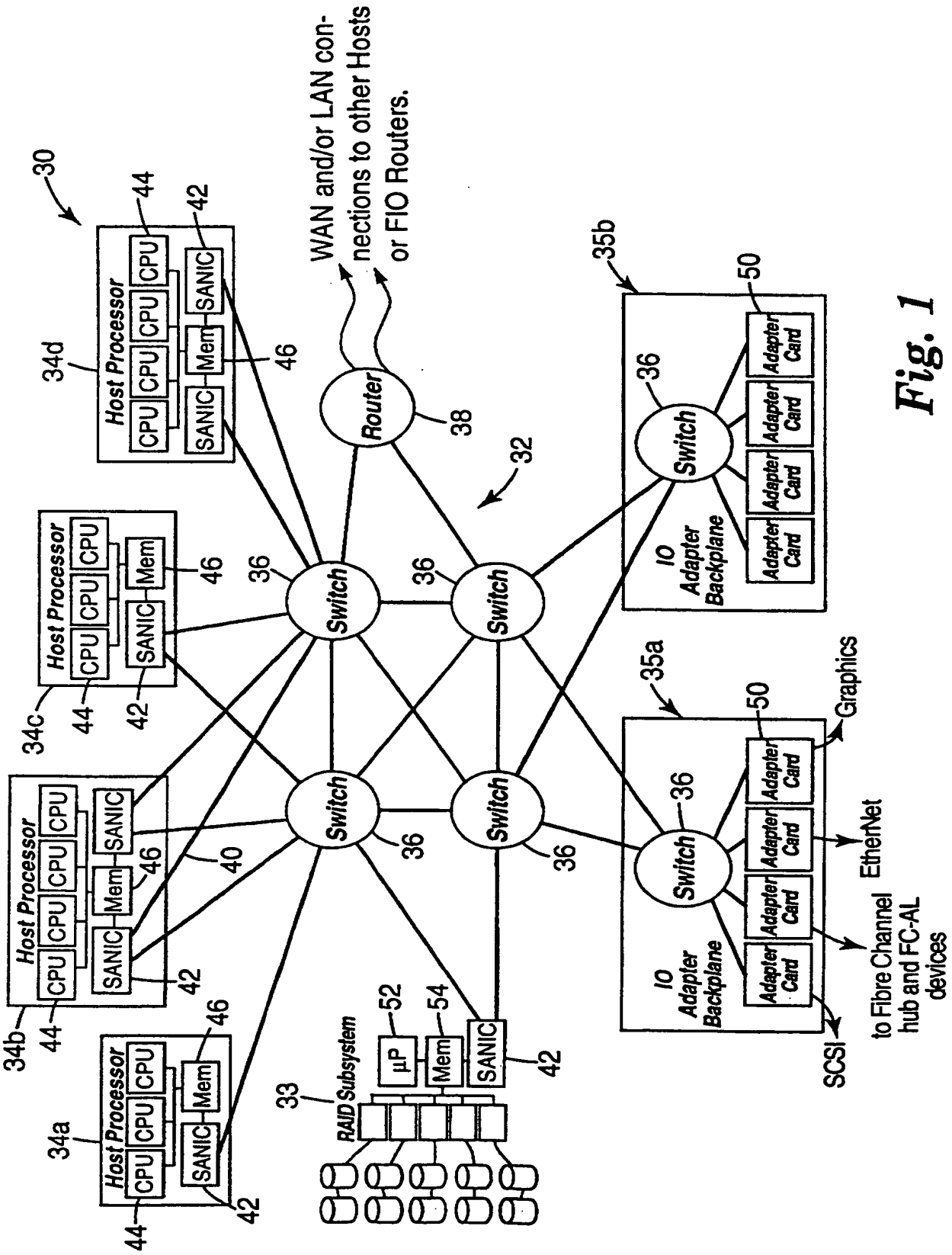

30
Host Processor
CPU
SANIC
Mem
34d
34c
34b
34a
44
42
46
WAN and/or LAN connections to other Hosts or FIO Routers.
Router
38
32
Switch
36
40
35b
35a
IO Adapter Backplane
Adapter Card
50
Graphics
EtherNet
to Fibre Channel hub and FC-AL devices
SCSI
RAID Subsystem
33
µP
52
54

Fig. 1

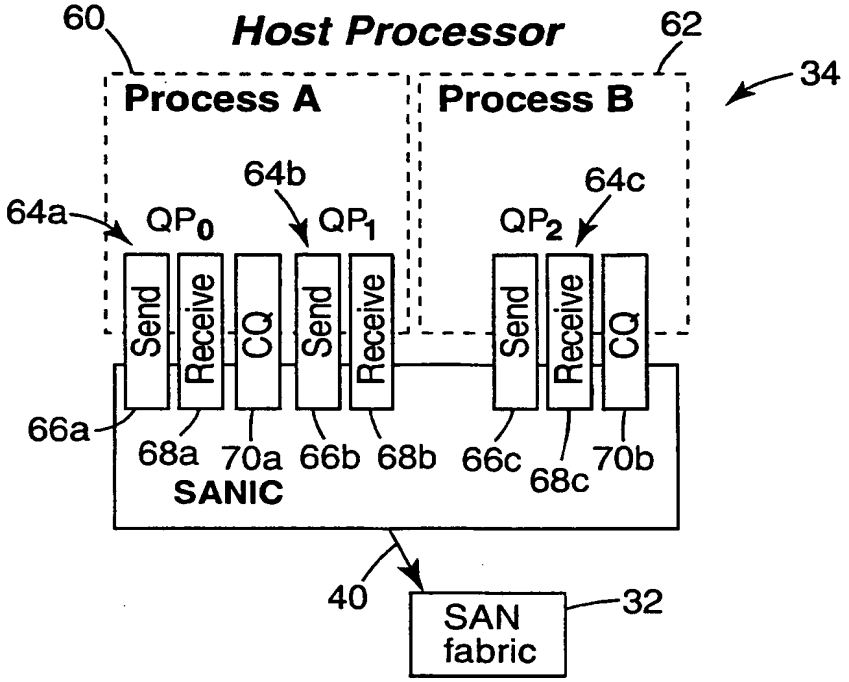

60
Host Processor
62
34
Process A
Process B
64a
64b
64c
QP0
QP1
QP2
Send
Receive
CQ
Send
Receive
Send
Receive
CQ
66a
68a
70a
66b
68b
66c
68c
70b
SANIC
40
SAN fabric
32

Fig. 2

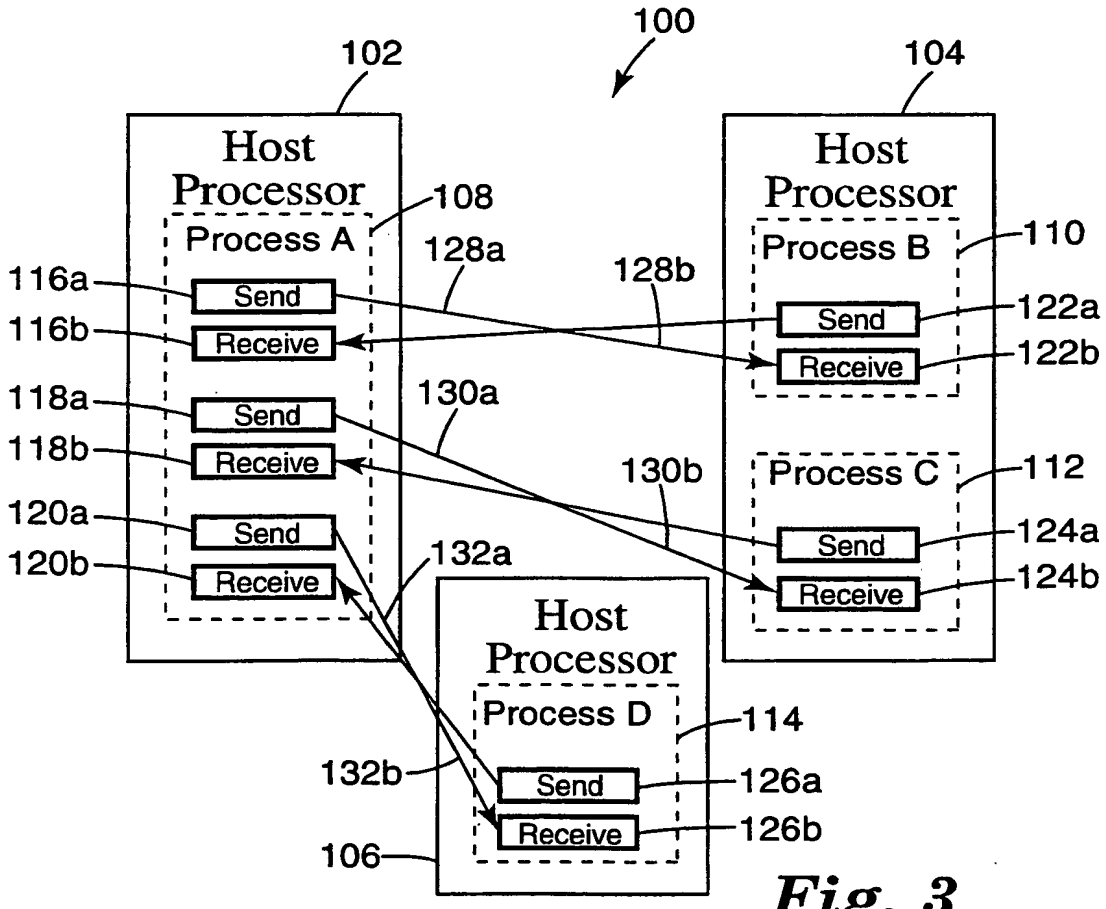

100
102
104
Host Processor
Process A
108
Host Processor
Process B
110
128a
128b
116a
116b
Send
Receive
Send
Receive
122a
122b
118a
118b
Send
Receive
130a
130b
Process C
112
120a
120b
Send
Receive
Send
Receive
124a
124b
132a
Host Processor
Process D
114
132b
Send
Receive
126a
126b
106

Fig. 3

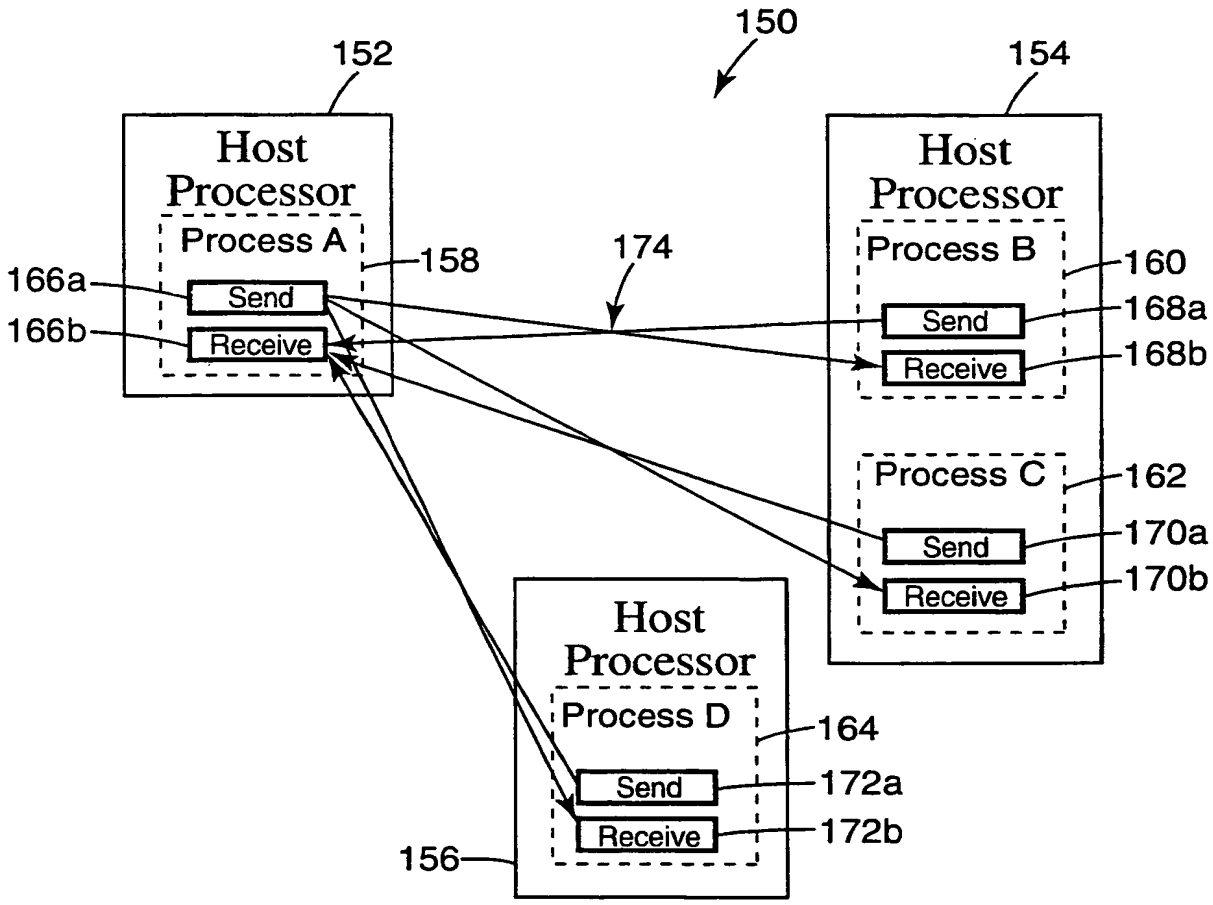

150
152
154
Host Processor
Process A
158
174
Process B
160
166a
Send
166b
Receive
168a
Send
168b
Receive
Process C
162
Send
170a
Receive
170b
Host Processor
Process D
164
Send
172a
Receive
172b
156

Fig. 4

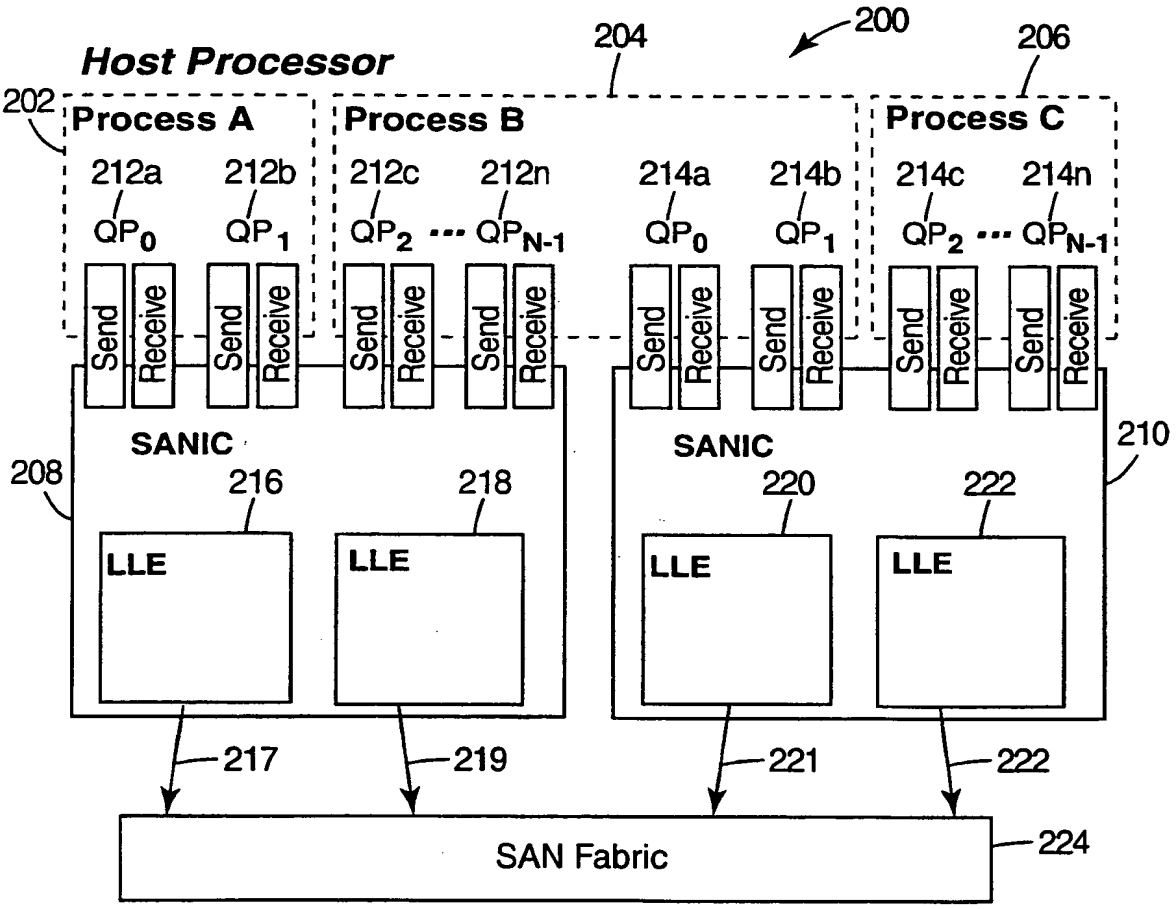

Host Processor
200
202
204
206
Process A
Process B
Process C
212a
212b
212c
212n
214a
214b
214c
214n
QP0
QP1
QP2
QPN-1
Send
Receive
SANIC
208
210
216
218
220
222
LLE
217
219
221
222
SAN Fabric
224

Fig. 5

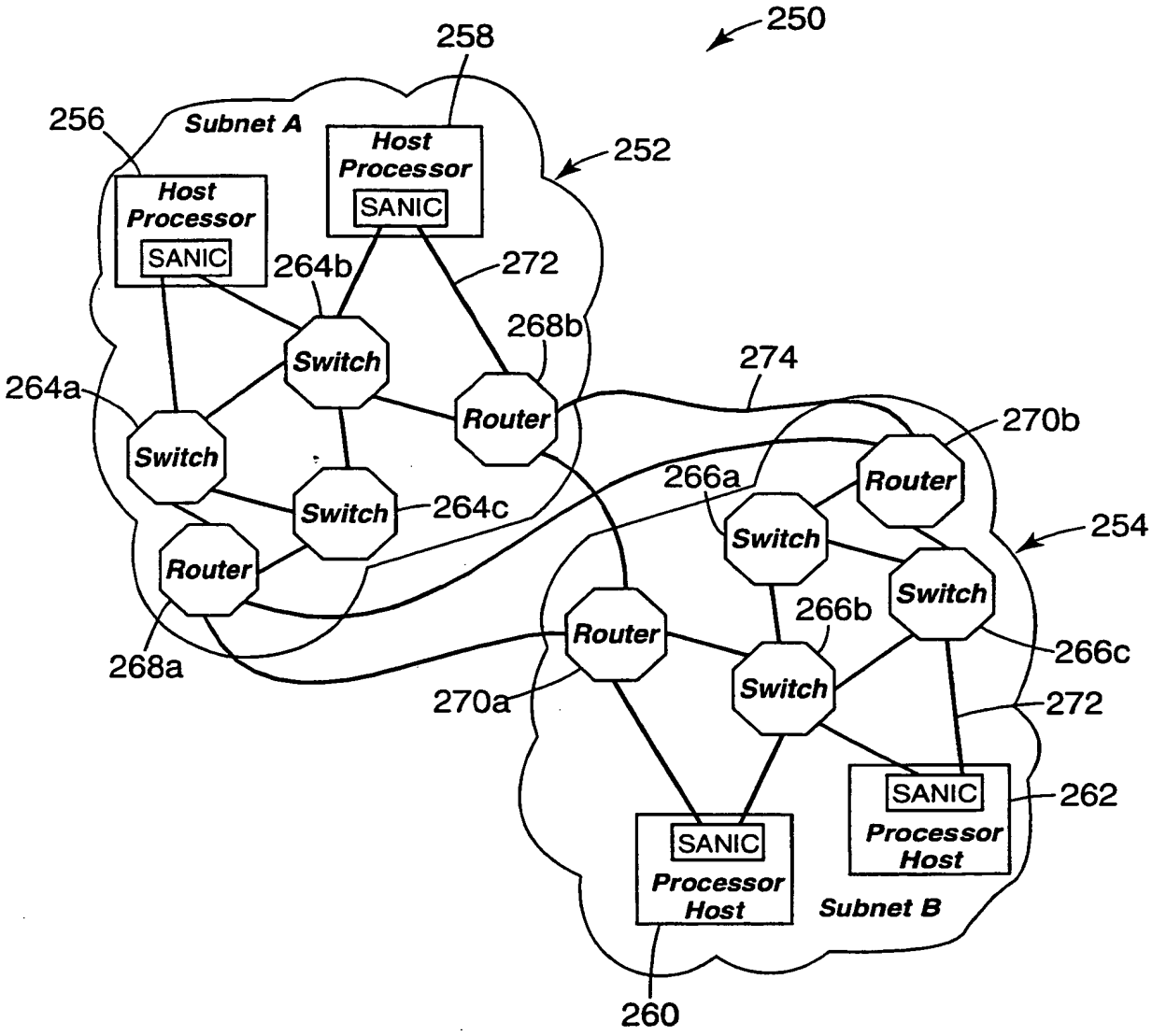

250
258
256
Subnet A
Host Processor
SANIC
252
Host Processor
SANIC
264b
272
268b
274
Switch
264a
Router
270b
Switch
266a
Router
Switch
264c
254
Router
Switch
266b
Switch
Router
266c
268a
270a
Switch
272
SANIC
262
Processor Host
SANIC
Processor Host
Subnet B
260

Fig. 6

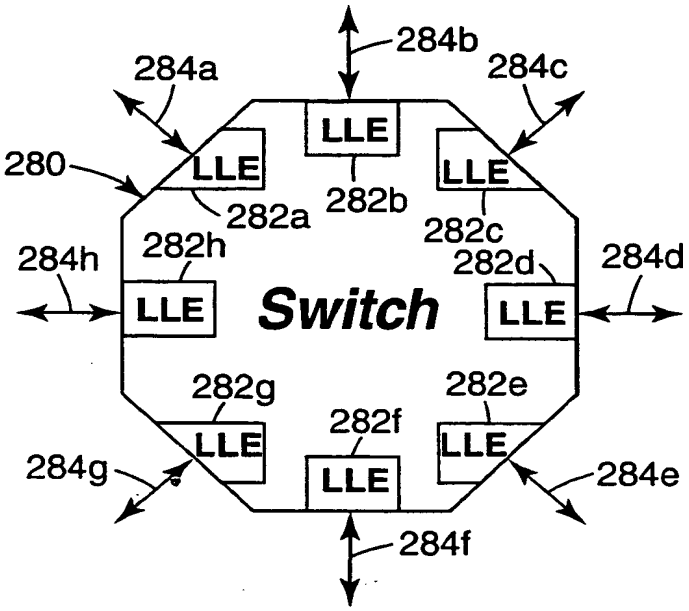

284b
284a
284c
280
LLE
LLE
LLE
282a
282b
282c
284h
282h
282d
284d
LLE
Switch
LLE
282g
282e
282f
LLE
LLE
LLE
284g
284e
284f

Fig. 7

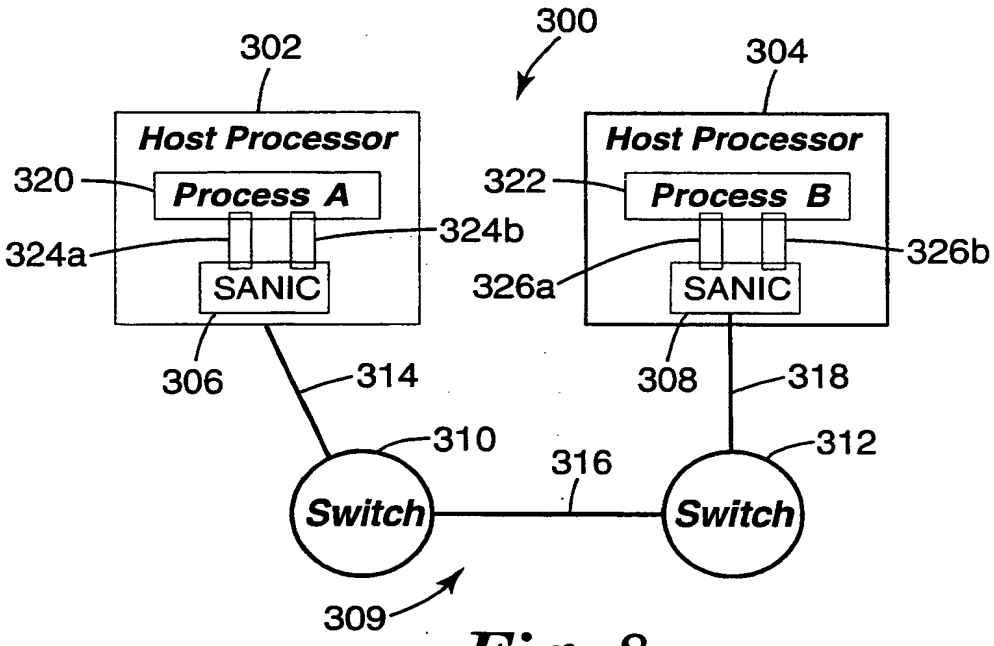

300
302
304
Host Processor
Host Processor
320
Process A
322
Process B
324a
324b
326a
326b
SANIC
SANIC
306
314
308
318
310
316
312
Switch
Switch
309

Fig. 8

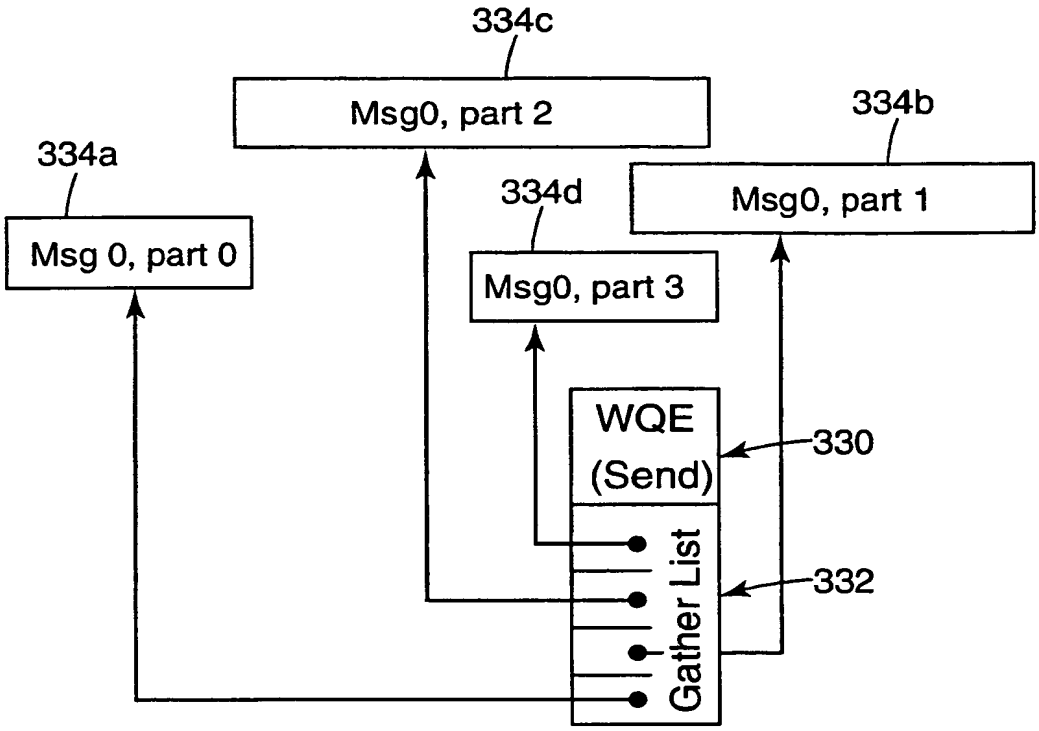

334c
Msg0, part 2
334b
334a
Msg0, part 1
334d
Msg 0, part 0
Msg0, part 3
WQE
(Send)
330
Gather List
332

Fig. 9A

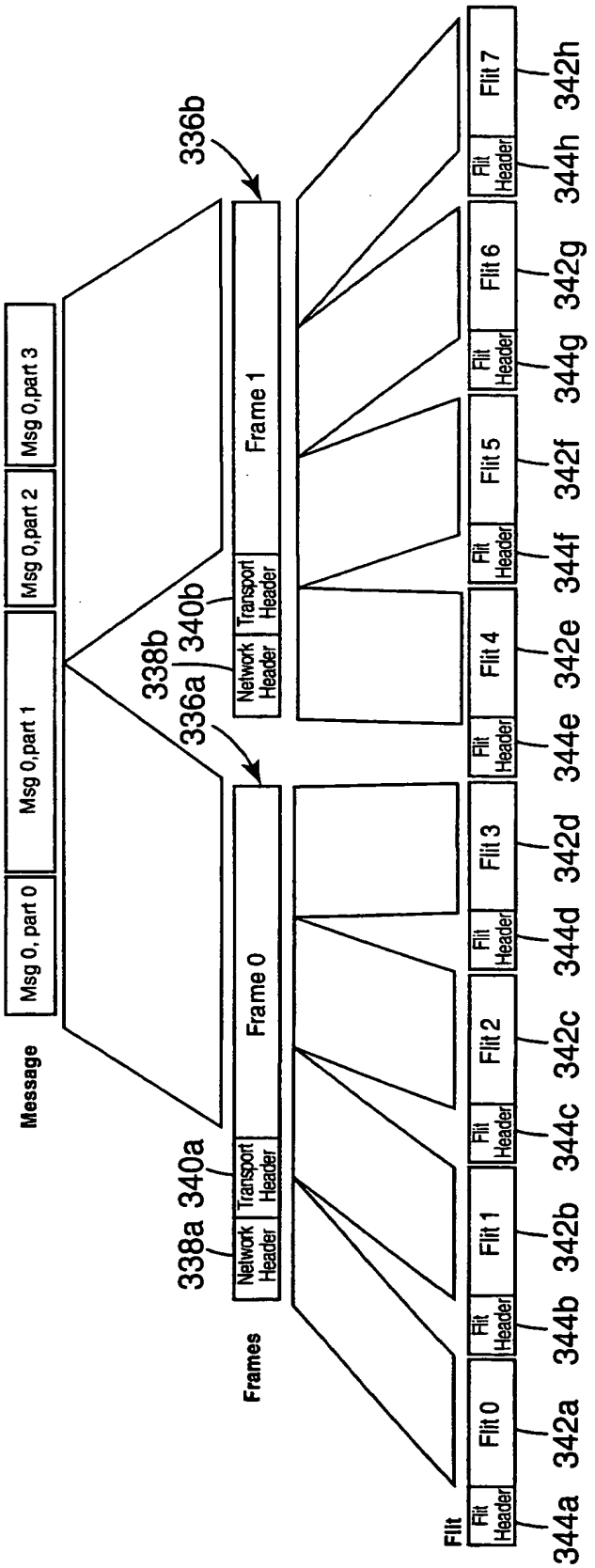

Message
Msg 0, part 0
Msg 0,part 1
Msg 0,part 2
Msg 0,part 3
336b
338b
336a
340b
338a
340a
Frames
Frame 0
Frame 1
Network Header
Transport Header
Flit
Flit Header
Flit 0
Flit 1
Flit 2
Flit 3
Flit 4
Flit 5
Flit 6
Flit 7
342a
342b
342c
342d
342e
342f
342g
342h
344a
344b
344c
344d
344e
344f
344g
344h

Fig. 9B

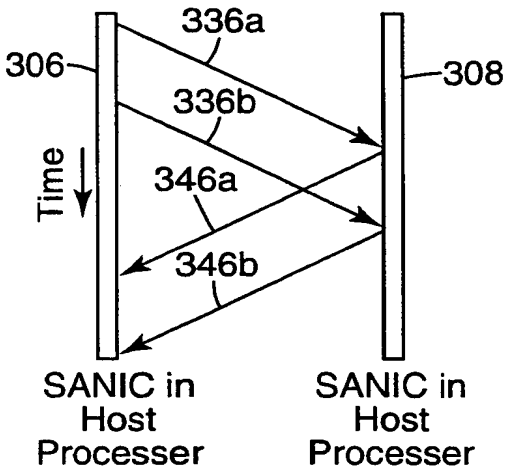


Fig. 10A

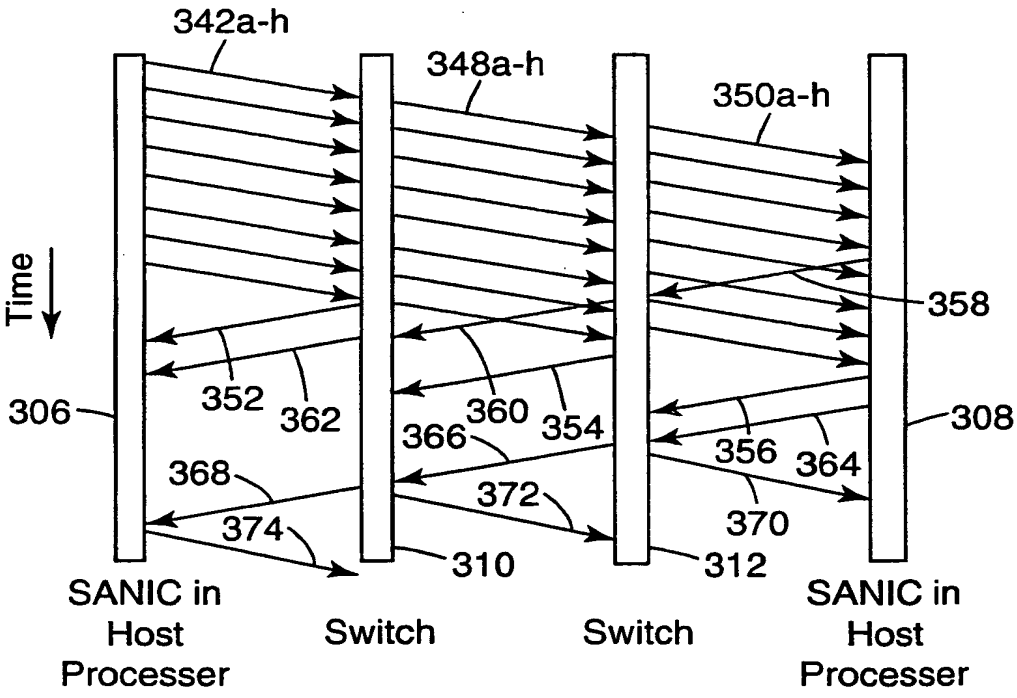


Fig. 10B

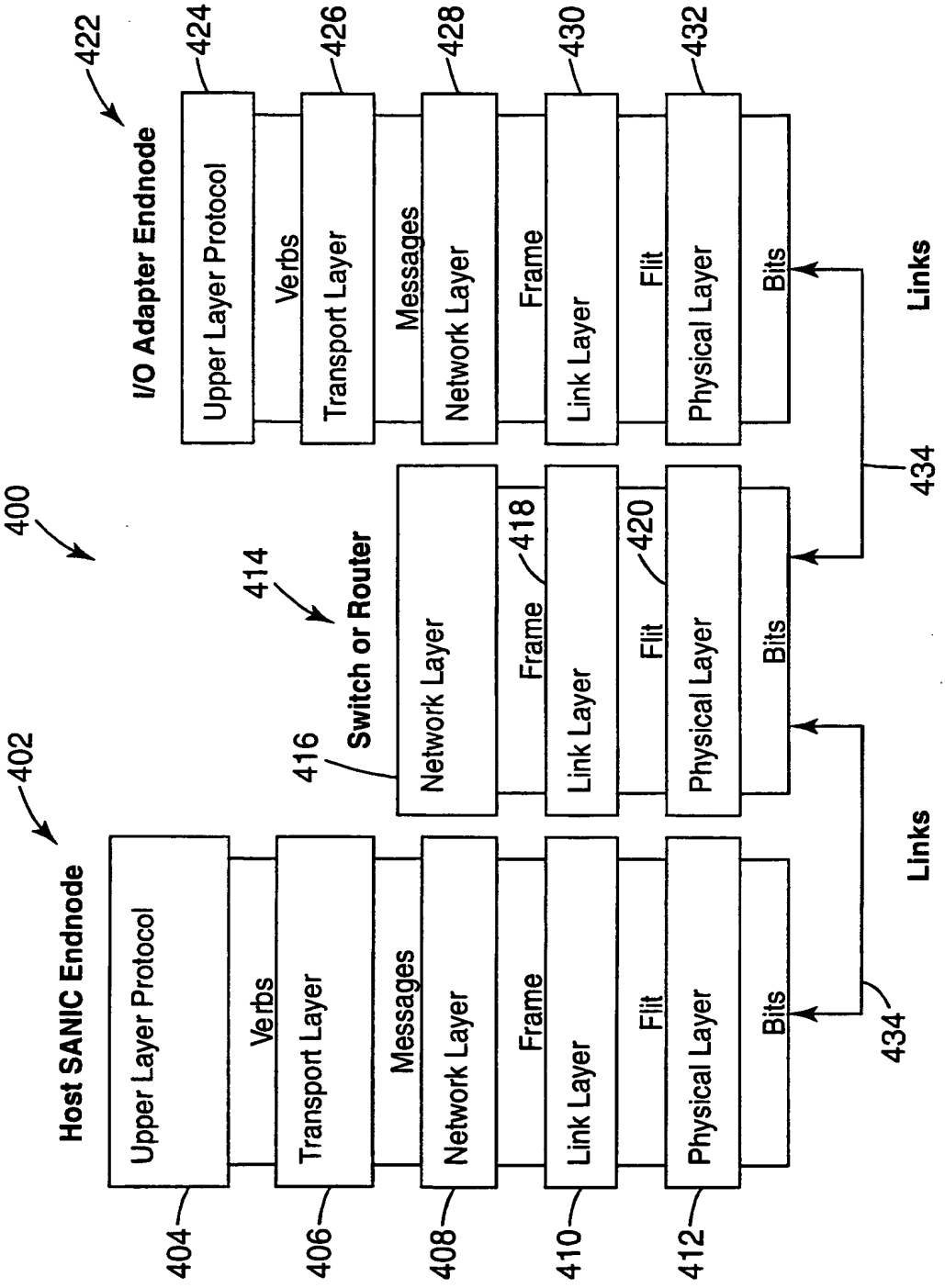

400
402
Host SANIC Endnode
Upper Layer Protocol
Verbs
Transport Layer
Messages
Network Layer
Frame
Link Layer
Flit
Physical Layer
Bits
404
406
408
410
412
414
Switch or Router
416
418
420
422
I/O Adapter Endnode
424
426
428
430
432
434
Links

Fig. 11

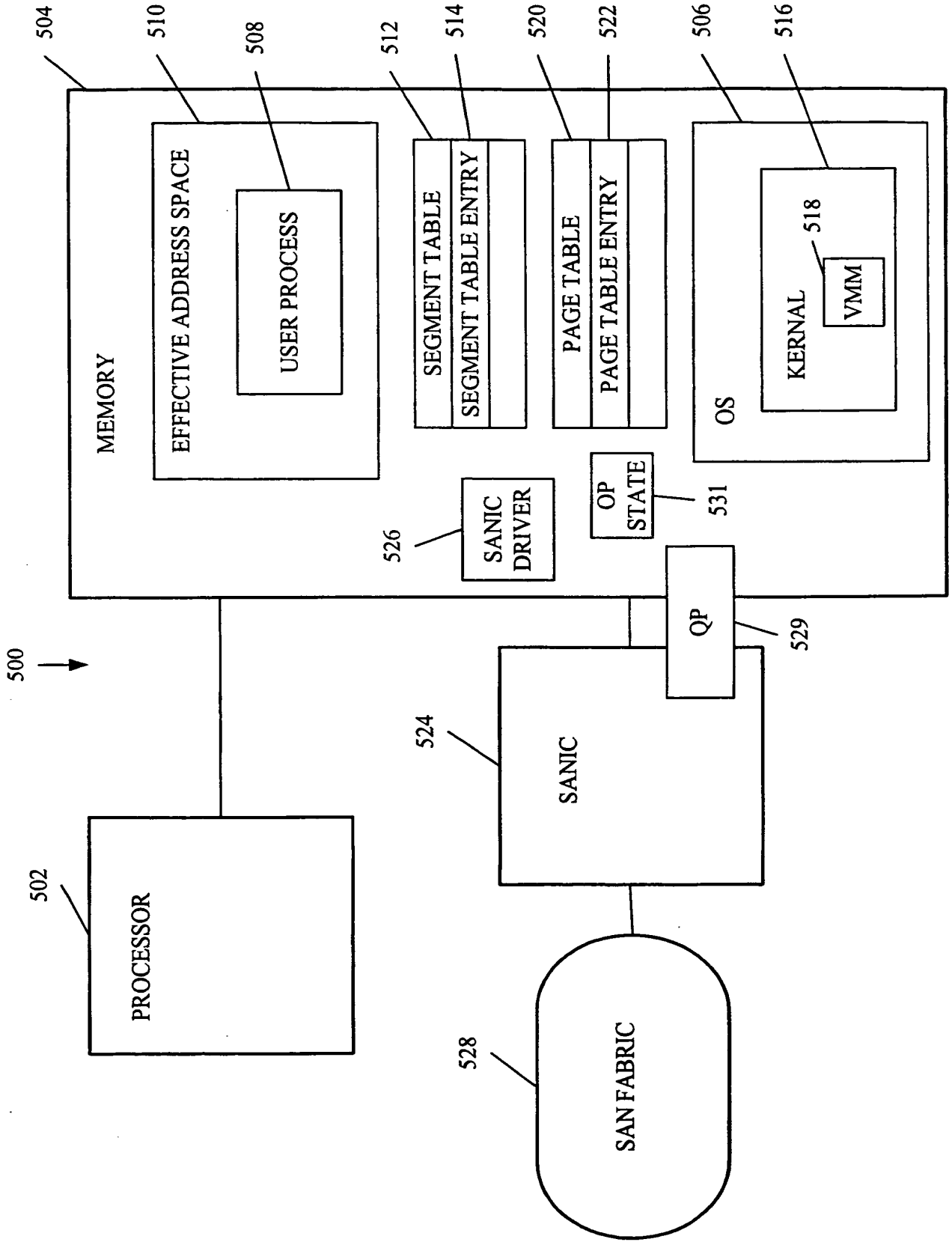

504
510
508
512
514
520
522
506
516
MEMORY
EFFECTIVE ADDRESS SPACE
USER PROCESS
SEGMENT TABLE
SEGMENT TABLE ENTRY
PAGE TABLE
PAGE TABLE ENTRY
OS
KERNAL
518
VMM
526
SANIC DRIVER
OP STATE
531
QP
529
500
524
SANIC
502
PROCESSOR
528
SAN FABRIC

FIG. 12

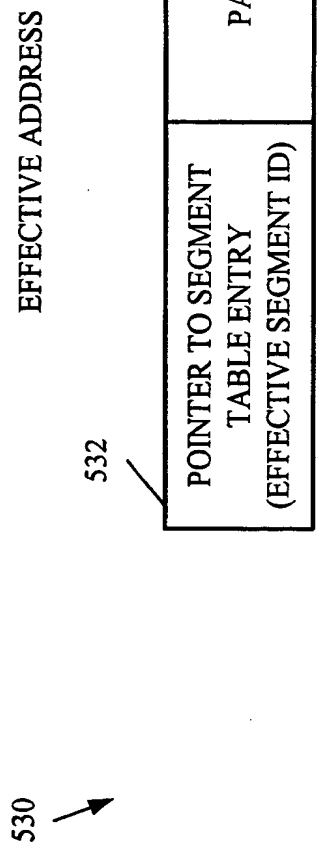

530
EFFECTIVE ADDRESS
532
POINTER TO SEGMENT TABLE ENTRY (EFFECTIVE SEGMENT ID)
534
PAGE NUMBER
536
PAGE OFFSET

FIG. 13

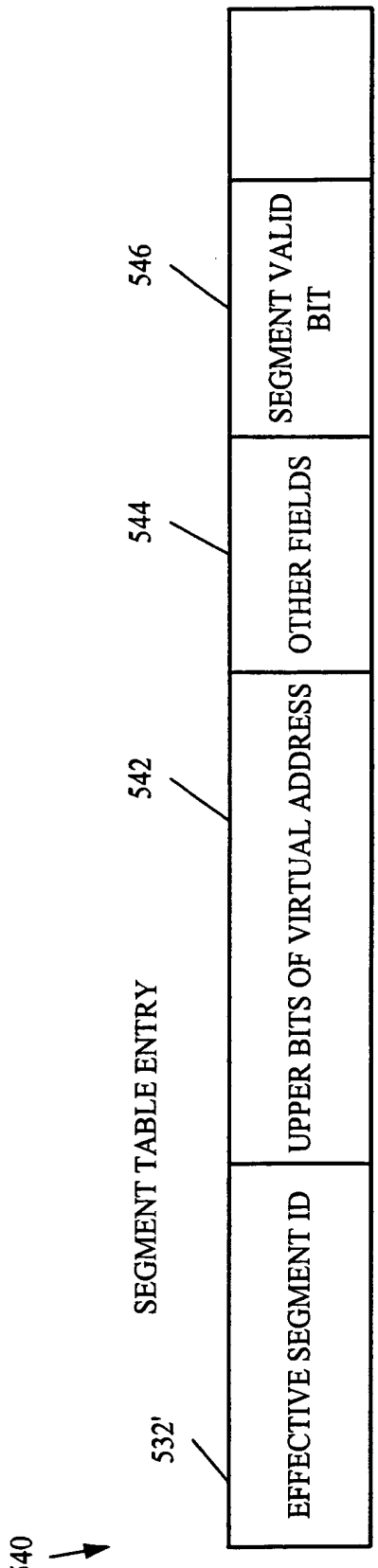

540
SEGMENT TABLE ENTRY
532'
EFFECTIVE SEGMENT ID
542
UPPER BITS OF VIRTUAL ADDRESS
544
OTHER FIELDS
546
SEGMENT VALID BIT

FIG. 14

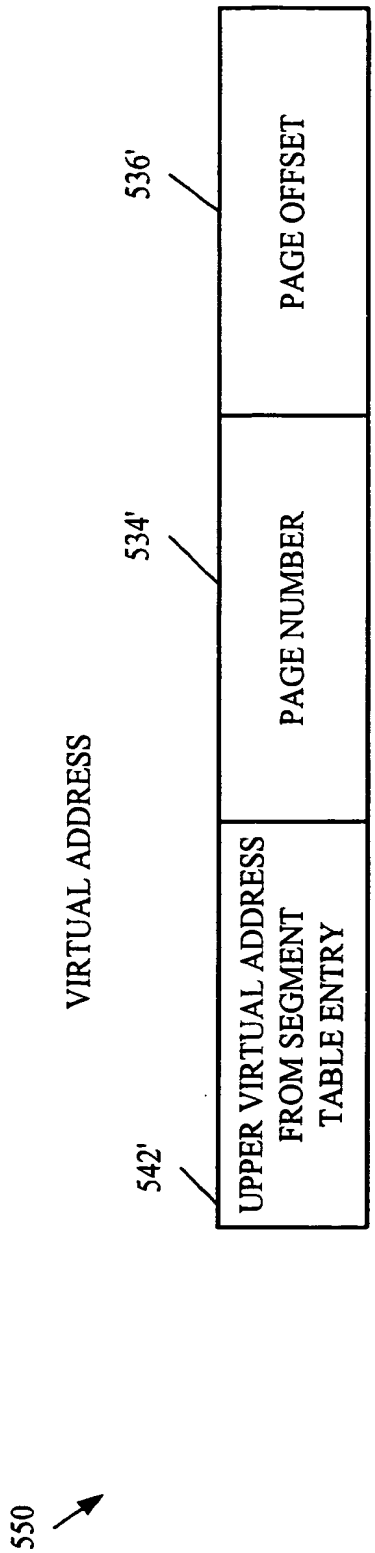

550
VIRTUAL ADDRESS
542'
UPPER VIRTUAL ADDRESS FROM SEGMENT TABLE ENTRY
534'
PAGE NUMBER
536'
PAGE OFFSET

FIG. 15

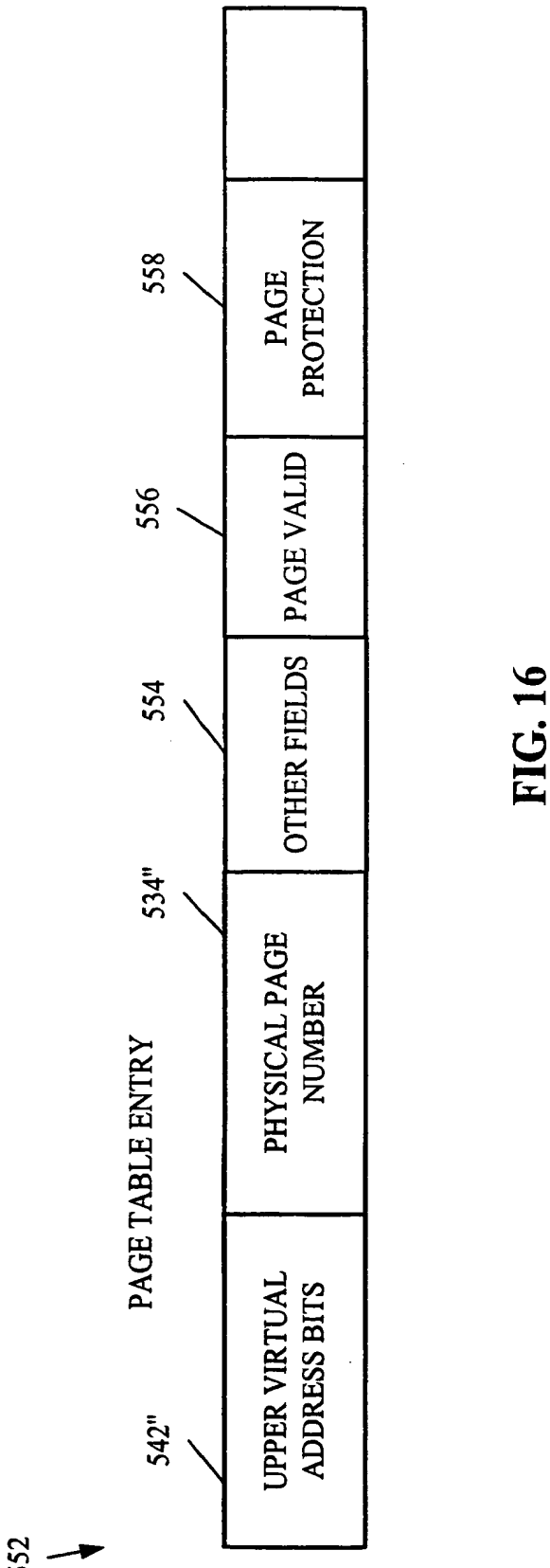

552
PAGE TABLE ENTRY
542"
UPPER VIRTUAL ADDRESS BITS
534"
PHYSICAL PAGE NUMBER
554
OTHER FIELDS
556
PAGE VALID
558
PAGE PROTECTION

FIG. 16

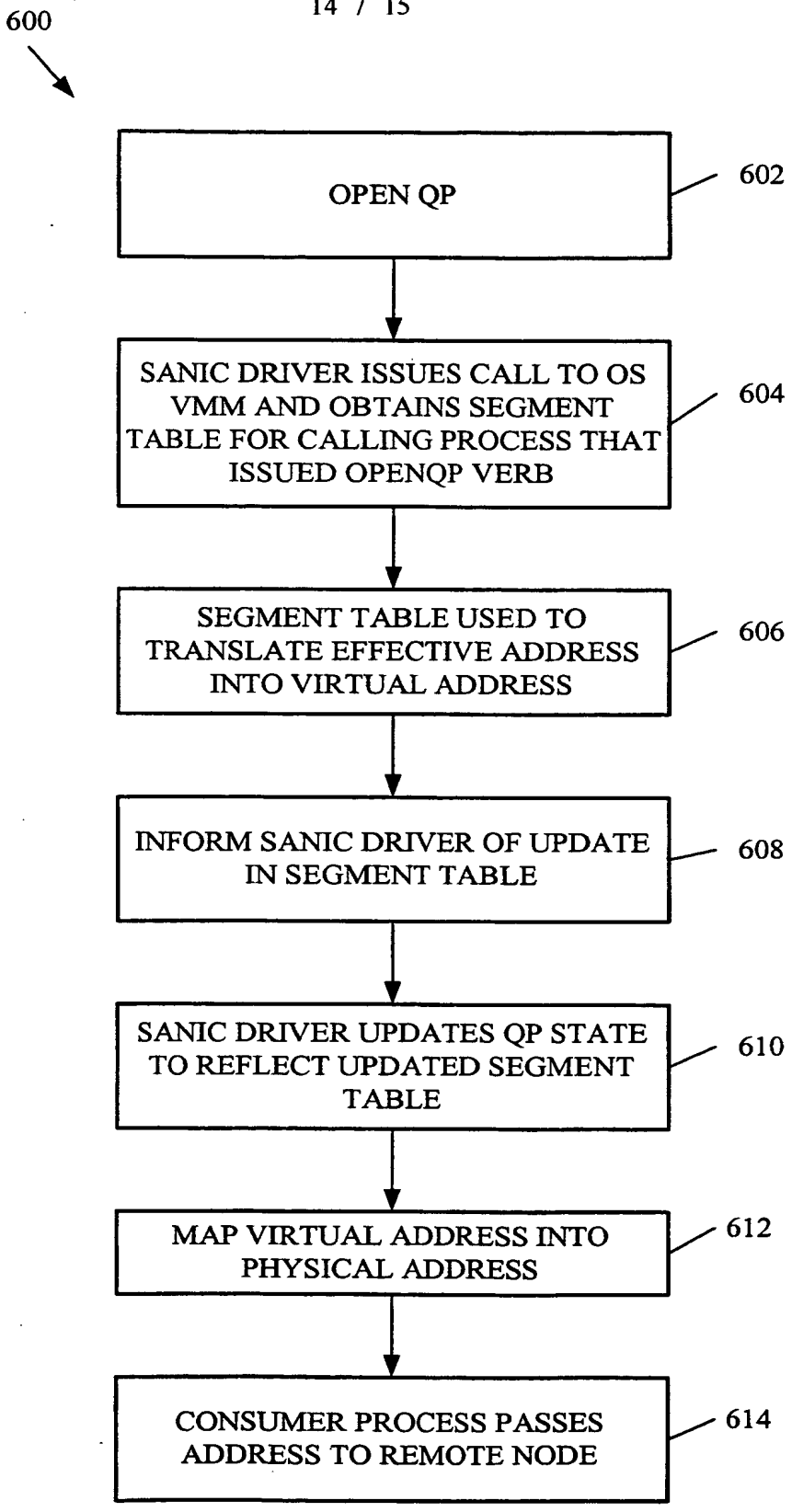

600
OPEN QP
602
SANIC DRIVER ISSUES CALL TO OS VMM AND OBTAINS SEGMENT TABLE FOR CALLING PROCESS THAT ISSUED OPENQP VERB
604
SEGMENT TABLE USED TO TRANSLATE EFFECTIVE ADDRESS INTO VIRTUAL ADDRESS
606
INFORM SANIC DRIVER OF UPDATE IN SEGMENT TABLE
608
SANIC DRIVER UPDATES QP STATE TO REFLECT UPDATED SEGMENT TABLE
610
MAP VIRTUAL ADDRESS INTO PHYSICAL ADDRESS
612
CONSUMER PROCESS PASSES ADDRESS TO REMOTE NODE
614

FIG. 17

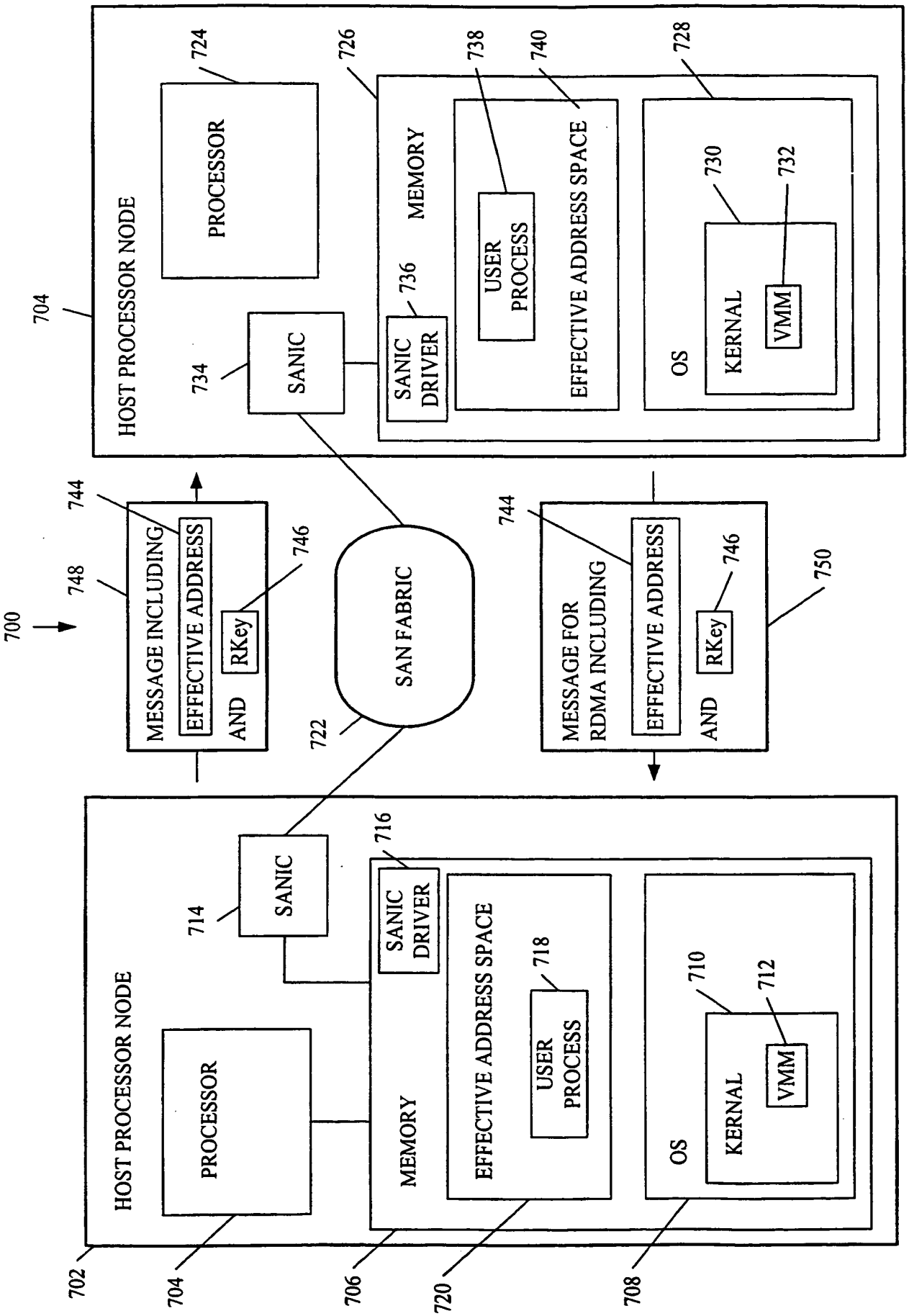

700
702
704
706
708
710
712
714
716
718
720
722
724
726
728
730
732
734
736
738
740
744
746
748
750
HOST PROCESSOR NODE
PROCESSOR
SANIC
SANIC DRIVER
MEMORY
EFFECTIVE ADDRESS SPACE
USER PROCESS
OS
KERNAL
VMM
SAN FABRIC
MESSAGE INCLUDING
EFFECTIVE ADDRESS
AND
RKey
MESSAGE FOR RDMA INCLUDING

FIG. 18

MA, MD, MG, MK, MN, MW, MX, MZ, NO, NZ, PL, PT, RO, RU, SD, SE, SG, SI, SK, SL, TJ, TM, TR, TT, TZ, UA, UG, US, UZ, VN, YU, ZA, ZW.

(84) **Designated States** *(regional)*: ARIPO patent (GH, GM, KE, LS, MW, MZ, SD, SL, SZ, TZ, UG, ZW), Eurasian patent (AM, AZ, BY, KG, KZ, MD, RU, TJ, TM), European patent (AT, BE, CH, CY, DE, DK, ES, FI, FR, GB, GR, IE, IT, LU, MC, NL, PT, SE), OAPI patent (BF, BJ, CF, CG, CI, CM, GA, GN, GW, ML, MR, NE, SN, TD, TG).

Published:
— *With international search report.*

For two-letter codes and other abbreviations, refer to the "Guidance Notes on Codes and Abbreviations" appearing at the beginning of each regular issue of the PCT Gazette.

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/14282

A. CLASSIFICATION OF SUBJECT MATTER

IPC(7) :G06F 15/167
US CL :709/212

According to International Patent Classification (IPC) or to both national classification and IPC

B. FIELDS SEARCHED

Minimum documentation searched (classification system followed by classification symbols)

U.S. : 709/212, 213, 214, 215, 216; 710/22, 26, 27

Documentation searched other than minimum documentation to the extent that such documents are included in the fields searched

Electronic data base consulted during the international search (name of data base and, where practicable, search terms used)

Please See Extra Sheet.

C. DOCUMENTS CONSIDERED TO BE RELEVANT

Category*	Citation of document, with indication, where appropriate, of the relevant passages	Relevant to claim No.
X, P	US 6,026,448 A (GOLDRIAN et al.) 15 FEBRUARY 2000 Figure 4, Column 1, Lines 16-61, Column 3, Line 48 through Column 5, Line 4, Column 6, Lines 33-63, Column 8, Lines 27-39, Column 10, Line 55 through Column 12, Line 19, Claim 1.	1
A	"Direct Memory Access TAG Architecture", IBM Technical Disclosure Bulletin, SEPTEMBER 1989, Vol 32, No. 4B, pages 143-151	1

☐ Further documents are listed in the continuation of Box C. ☐ See patent family annex.

*	Special categories of cited documents:	"T"	later document published after the international filing date or priority date and not in conflict with the application but cited to understand the principle or theory underlying the invention
"A"	document defining the general state of the art which is not considered to be of particular relevance	"X"	document of particular relevance; the claimed invention cannot be considered novel or cannot be considered to involve an inventive step when the document is taken alone
"E"	earlier document published on or after the international filing date	"Y"	document of particular relevance; the claimed invention cannot be considered to involve an inventive step when the document is combined with one or more other such documents, such combination being obvious to a person skilled in the art
"L"	document which may throw doubts on priority claim(s) or which is cited to establish the publication date of another citation or other special reason (as specified)	"&"	document member of the same patent family
"O"	document referring to an oral disclosure, use, exhibition or other means		
"P"	document published prior to the international filing date but later than the priority date claimed		

Date of the actual completion of the international search 27 JULY 2000	Date of mailing of the international search report 22 AUG 2000
Name and mailing address of the ISA/US Commissioner of Patents and Trademarks Box PCT Washington, D.C. 20231 Facsimile No. (703) 305-3230	Authorized officer MARC THOMPSON Telephone No. (703) 305-3900

Form PCT/ISA/210 (second sheet) (July 1998) ★

INTERNATIONAL SEARCH REPORT

International application No.
PCT/US00/14282

B. FIELDS SEARCHED
Electronic data bases consulted (Name of data base and where practicable terms used):

EAST patent full text search, IBM Technical Disclosure Bulletins, IEEE database search
Keywords: Direct Memory Access (DMA), Remote DMA (RDMA), DMA Control (DMAC), memory range, binding, key/tag/ID.